

Les transferts d'apprentissage dans le cadre des transferts technologiques informatiques : le cas du maquettage en conception informatique

Marc-Eric Bobillier Chaumon

▶ **To cite this version:**

Marc-Eric Bobillier Chaumon. Les transferts d'apprentissage dans le cadre des transferts technologiques informatiques : le cas du maquettage en conception informatique. Ordinateur et société [cs.CY]. Université Paul Verlaine - Metz, 1999. Français. ⟨NNT : 1999METZ004L⟩. ⟨tel-01775782⟩

HAL Id: tel-01775782

https://hal.univ-lorraine.fr/tel-01775782v1

Submitted on 24 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

THÈSE

présentée à

L'UNIVERSITÉ DE METZ

discipline :

PSYCHOLOGIE

pour obtenir :

LE DIPLÔME DE DOCTEUR

par :

MARC-ERIC BOBILLIER CHAUMON

Sujet de la thèse

Les transferts d'apprentissage dans le cadre des transferts technologiques informatiques : Le cas du maquettage en conception informatique

Sous la direction du Professeur G.N. FISCHER

Soutenue le 5 janvier 1999

BIBLIOTHEQUE UNIVERSITAIRE DE METZ
031 291940 9

Membres du Jury :

M. M. BELLI	Informatique-CDC	(Examinateur)
M. E. BRANGIER	Maître de Conférence à l'Université de Metz	(Examinateur)
M. G.N. FISCHER	Professeur à l'Université de Metz	(Examinateur)
M. G. MINGUET	Professeur à l'École des Mines de Nantes	(Examinateur)
M. D. L. SCAPIN	Directeur de Recherche à l'INRIA	(Rapporteur)
M. C. A. TIJUS	Professeur à l'Université de Paris VIII	(Rapporteur)

BIBLIOTHEQUE UNIVERSITAIRE
LETTRES - METZ
N° inv. 19990251
Cote L/M3 99/4
Loc MAGASIN

THÈSE

présentée à

L'UNIVERSITÉ DE METZ

discipline :

PSYCHOLOGIE

pour obtenir :

LE DIPLÔME DE DOCTEUR

par :

MARC-ERIC BOBILLIER CHAUMON

Sujet de la thèse

Les transferts d'apprentissage dans le cadre des transferts technologiques informatiques : Le cas du maquettage en conception informatique

Sous la direction du Professeur G.N. FISCHER

Soutenue le 5 janvier 1999

Membres du Jury :

M. M. BELLI	Informatique-CDC	(Examinateur)
M. E. BRANGIER	Maître de Conférence à l'Université de Metz	(Examinateur)
M. G.N. FISCHER	Professeur à l'Université de Metz	(Examinateur)
M. G. MINGUET	Professeur à l'École des Mines de Nantes	(Examinateur)
M. D. L. SCAPIN	Directeur de Recherche à l'INRIA	(Rapporteur)
M. C. A. TIJUS	Professeur à l'Université de Paris VIII	(Rapporteur)

Remerciements

Ce travail est le fruit d'une collaboration féconde avec de nombreuses personnes qui ont toujours répondu favorablement à mes nombreuses sollicitations.

Je remercie vivement les membres du jury de cette thèse :

- Monsieur Gustave Nicolas Fischer, Professeur à l'Université de Metz, mon directeur de thèse, pour son soutien, sa compréhension et ses nombreux conseils. Je lui suis reconnaissant de la précision de sa relecture et de la qualité de ses remarques, de forme comme de fond, témoignant d'une grand valeur scientifique.
- Monsieur Eric Brangier, Maître de Conférence à l'Université de Metz, mon co-directeur de thèse, qui a initié, supervisé et finalisé ce travail de recherche. Il a su guider mes travaux tout en préservant les applications pratiques indispensables à la société Informatique-CDC. Il a contribué efficacement au développement des idées nouvelles ainsi qu'à la présentation de cette thèse. Qu'il reçoive ici l'expression de ma profonde reconnaissance pour sa persévérance qui n'a jamais faibli à suivre ce projet.
- Monsieur Charles Tijus, Professeur à l'Université de Paris VIII, qui me fait l'honneur d'être membre de ce jury. Son écoute attentive et ses remarques constructives m'ont aidé aux tournants importants de mes recherches.
- Monsieur Dominique Scapin, Directeur de Recherche à l'INRIA, pour le plaisir et l'honneur qu'il me fait en acceptant d'être le rapporteur de cette thèse. Qu'il soit assuré de l'expression de ma profonde gratitude pour cette participation.
- Monsieur Guy Minguet, Professeur à l'École des Mines de Nantes, pour avoir eu l'amabilité de participer au jury de cette thèse. Qu'il trouve ici l'expression de mes sentiments reconnaissants.
- Monsieur Michel Belli, Responsable du service Méthodes Ingénierie et Progiciels d'Informatique-CDC, qui m'a accueilli au sein de son service, en me procurant d'excellentes conditions de travail. Il a su trouver des applications nouvelles pour la société tout en m'encourageant à explorer les aspects théoriques nécessaires au travail de recherche. Il m'a ainsi permis de conserver dans mes interventions un équilibre délicat mais fécond.

Je remercie chaleureusement toutes les personnes du groupe Informatique-CDC qui, de près ou de loin, ont contribué à ce travail. Mes remerciements s'adressent plus particulièrement à l'ensemble du personnel du service DABFI qui m'ont donné la possibilité de réaliser mes recherches dans un cadre humain d'une rare qualité.

Je remercie également Monsieur Jean-Marie SEZERAT, responsable du service Recherche Développement Techniques Avancées d'Informatique-CDC qui, au cours de la première période de ce travail, a contribué à son bon déroulement. Il a accueilli mon initiative avec enthousiasme et a su orienter mes premiers pas dans une grande société informatique.

Ces années de thèse se sont particulièrement bien déroulées grâce au soutien sans réserve et aux nombreux échanges avec mes collègues de la société Informatique-CDC, en particulier ceux de MIP et de RDT. Leurs compétences et leurs qualités humaines m'ont beaucoup apporté. Il serait trop long de les citer, mais tous sauront se reconnaître. Tout en me laissant une grande liberté d'action, ils ont su m'aider, m'encourager et me faire bénéficier de leurs profondes connaissances du monde de l'informatique et des nouvelles technologies.

Je suis également très reconnaissant aux membres du Département de Psychologie de l'Université de Metz de m'avoir accompagné et conseillé pendant ces dernières années de thèse. En me faisant bénéficier de leur expérience et de leurs compétences, en étant à l'écoute de mes doutes et de mes interrogations, en répondant favorablement à mes sollicitations, ils ont su créer un environnement de travail amical et très fructueux. Leur bonne humeur et leurs remarques toujours très pertinentes ont contribué au bon déroulement de cette thèse.

Je tiens également à exprimer toute ma gratitude aux personnes qui ont bien voulu relire les épreuves de mon manuscrit. Merci donc à : Nadjma et Daniel (Namibie), Claude (Athènes), Hélène (Bordeaux), Jean-Marc (Belfort), Stéphanie, Frantz, Francis, Sylvain, Timo, Jean-Luc et Michel (Paris), Anna-Maria, Cyril, Javier et Loïc (Metz).

Cette recherche fut possible grâce au soutien de l'Association Nationale de la Recherche Technique, ANRT, qui dans le cadre d'une Convention Industrielle de Formation et de Recherche, CIFRE, a financé une partie de cette étude. Je tiens personnellement à l'en remercier.

Le soutien de Monsieur Hervé Stoppiglia, ancien condisciple thésard d'I-CDC, m'a été très utile pour achever ce travail.

Je remercie Delphine, mon amie, pour son soutien sans faille, ses encouragements renouvelés et sa foi inébranlable en l'issue favorable de cette recherche.

Enfin, à tous ceux, parents, frère et amis, qui dans les moments de doute, par leur amitié et leur affection, m'ont encouragé et soutenu, j'adresse ici mes remerciements les plus chaleureux.

Résumé

Cette recherche se propose d'étudier, en situation réelle de maquettage, les transferts de compétences chez des informaticiens confrontés à l'évolution technologique de leur environnement de conception : passage d'un atelier de génie logiciel (Pacbase) sur site-central vers un langage de quatrième génération (NSDK) en client serveur.

La méthode utilisée pour déterminer ces compétences consiste à dresser le modèle mental moyen de trois catégories d'informaticiens (Experts Pacbase ; Experts NSDK et Novices NSDK –mais ayant une expérience préalable en Pacbase–) et à les comparer entre eux. A partir des théories générales des schémas, du modèle mental, et du raisonnement analogique, nous avons élaboré un modèle d'analyse psycho-cognitif : (*i*) pour déterminer les raisonnements relatifs à chaque groupe, (*ii*) pour diagnostiquer et expliquer les types d'erreurs commises par les novices, (*iii*) pour proposer des recommandations favorisant la transition technologique. Concrètement, cette grille d'analyse a permis de paramétrer les modèles mentaux des individus selon quatre niveaux conceptuels : l'organisation de l'activité (modèle d'action) ; les stratégies de résolution des problèmes et de conception de la maquette (modèle de résolution) ; les modalités de l'interaction homme-machine (modèle de l'interaction) et les connaissances nécessaires à l'élaboration de la maquette (modèle de référence).

Les résultats, tant quantitatifs et qualitatifs, obtenus par l'observation et l'analyse détaillée des protocoles individuels tendent à démontrer que les problèmes rencontrés par ces novices pour concevoir et interagir avec le nouvel environnement technique proviendraient principalement de l'incompatibilité des compétences acquises en site-central (et réutilisées via des raisonnements analogiques) avec celles requises pour développer en client serveur. Outre ces transferts négatifs, le système technique source est également incriminé car il incite l'informaticien à adopter un mode de fonctionnement passablement rigide et sclérosé, qui l'empêche de faire évoluer ses raisonnements vers des modèles plus appropriés au domaine cible. Un modèle explicatif de ces dysfonctionnements cognitifs est d'ailleurs développé. Des recommandations ergonomiques sont enfin proposées pour favoriser la transition technologique.

Mots Clefs : Ergonomie cognitive, Psychologie ergonomique de la programmation, Transfert d'apprentissage, Transfert technologique, Conception informatique, Interaction homme-machine.

SUMMARY

The goal of this research was to investigate, in the actual situation of designing a software prototype, the transfer of competencies in software designers who are confronted with the rapid technological advances of their working environments. In particular, we investigated the transfer of knowledge from a tool of procedural programming on a mainframe environment (Pacbase) to a new graphical design tool using a network and a workstation (NSDK).

The procedure used to identify the transfer of competencies involved determining the average mental model of three classes of software designers (those who are expert in Pacbase, those who are expert in NSDK, and the beginners in NSDK who had prior knowledge of Pacbase) and in comparing the performances of these groups.

Based on general framework theories about schema, mental models, and analogical reasoning, we developed a psycho-cognitif analysis model : (i) to determine the reasoning patterns of each group of software designers, (ii) to diagnose and explain the type of errors made by beginners, (iii) to propose recommendations aimed at improving the technological transfer. This model of analysis allowed us to classify the mental models of the software designers according to four conceptual levels : (i) the organization of their activity (action model), (ii) the different strategies used in problem solving and in prototype design (problem solving model), (iii) the types of human-machine interaction (interaction model), and (iv) the knowledge necessary to build the prototype (reference model).

Quantitative and qualitative data obtained both from observations and from the detailed analysis of the individual protocols indicated that the problems met by beginners to conceive the prototype and interact with the new environment are due essentially to the inadequacy between their previous competencies using a mainframe (applied by analogical reasoning to the new situation), and the new knowledge required to develop the prototype in the next programming environment. Besides this negative transfer, the old tool (Pacbase) is also to blame because it forces the software designers to adopt a strict and rigid mode of reasoning. So, the designers cannot adjust their reasoning to the new environment. An explanatory model of these cognitive disturbances has been developed. Finally, ergonomic recommendations are proposed to help the transition to a new technology.

Keywords : Cognitive ergonomics, Ergonomic psychology of programming, Transfer of competencies, Technological transfer, Software design, Human-machine interaction

SOMMAIRE

INTRODUCTION 1

CHAPITRE I
Le cadre de recherche : Aspects généraux de la conception Informatique et des transferts technologiques

1. INTRODUCTION 8

2. LA TÂCHE DE CONCEPTION INFORMATIQUE : DE QUOI PARLONS-NOUS ? 9

3. UNE APPROCHE HISTORIQUE DE LA CONCEPTION INFORMATIQUE 11

3.1 UNE ÉVOLUTION RAPIDE ET MOUVEMENTÉE 11
3.2 EN RÉSUMÉ 16

4. LES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE .. 17

4.1 TRANSFERT TECHNOLOGIQUE : DE QUOI S'AGIT-IL ? 17
4.2 LES IMPLICATIONS HUMAINES DES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE 19
4.3 LES IMPLICATIONS SOCIO-ORGANISATIONNELLES DES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE 21
4.4 EN RÉSUMÉ 22

5. ÉVOLUTION DES OUTILS ET DES MÉTHODES DE CONCEPTION INFORMATIQUE 23

5.1 LES DISPOSITIFS TECHNIQUES DE CONCEPTION INFORMATIQUE 23
5.2 LES DÉMARCHES DE CONDUITE DE PROJET EN CONCEPTION INFORMATIQUE 39

6. CONCLUSION 50

Chapitre II
Les aspects cognitifs de l'activité de conception informatique

1. INTRODUCTION 52

2. LE POINT DE VUE DE L'ERGONOMIE DE LA PROGRAMMATION 53

2.1 ERGONOMIE DES SYSTÈMES TECHNIQUES ET DE L'INFORMATIQUE 53
2.2 GENÈSE DE L'ERGONOMIE DE LA PROGRAMMATION 54
2.3 UN APERÇU DES DOMAINES D'INTERVENTION EN ERGONOMIE DE LA PROGRAMMATION 56
2.4 EN RÉSUMÉ 60

3. LES CARACTÉRISTIQUES COGNITIVES DE L'ACTIVITÉ DE CONCEPTION 61

3.1 UNE ACTIVITÉ DE RÉSOLUTION DE PROBLÈME 61
3.2 EN RÉSUMÉ 65

4. LES ÉTAPES COGNITIVES DE L'ACTIVITÉ DE CONCEPTION 66

4.1 LA COMPRÉHENSION DU PROBLÈME 66
4.2 LA CONSTRUCTION D'UNE SOLUTION 68
4.3 LA RÉUTILISATION DE SOLUTIONS 77

4.4 LES MODES DE PLANIFICATION 81
4.5 L'IMPLÉMENTATION DE LA SOLUTION RETENUE 86
4.6 EN RÉSUMÉ 91

5. LES TRANFERTS DE COMPÉTENCES DANS LA CONCEPTION 93

5.1 LE RAISONNEMENT ANALOGIQUE : DE QUOI S'AGIT-IL ? 93
5.2 LES DIFFÉRENTES FONCTIONS DU RAISONNEMENT ANALOGIQUE DANS LA CONCEPTION 94
5.3 LES PRINCIPES DE FONCTIONNEMENT DU RAISONNEMENT ANALOGIQUE 96
5.4 CARACTÉRISTIQUES DES COMPÉTENCES TRANSFÉRÉES 108

6. CONCLUSION 120

CHAPITRE III
Le travail de recherche sur le terrain

1. CADRE MÉTHODOLOGIQUE 124

1.1 PROBLÉMATIQUE ET HYPOTHÈSES 125
1.2 DÉMARCHE GÉNÉRALE D'INTERVENTION 136

2. LA DESCRIPTION DU SYSTÈME HOMME-MACHINE : LE CONTEXTE DE L'ANALYSE 146

2.1 LES CARACTÉRISTIQUES TECHNIQUES DES ENVIRONNEMENTS DE CONCEPTION ÉTUDIÉS 147
2.2 L'ORGANISATION DE L'ACTIVITÉ DE MAQUETTAGE 174
2.3 L'ANALYSE DES CONTRAINTES ET DES ASTREINTES 190
2.4 CONCLUSION GÉNÉRALE SUR L'ANALYSE DU SYSTÈME HOMME-MACHINE 204

3. L'ANALYSE COGNITIVE DE L'ACTIVITÉ DE MAQUETTAGE 207

3.1 LE MODÈLE D'ACTION 208
3.2 MODÈLE DE RÉSOLUTION 233
3.3 MODÈLE D'INTERACTION 271
3.4 MODÈLE DE RÉFÉRENCE 283

CHAPITRE IV
Discussion générale et Conclusion

1. DISCUSSION GÉNÉRALE 295

1.1 LES MODÈLES MENTAUX EN PRÉSENCE 295
1.2 PRINCIPALES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES 297
1.3 DEUX MODÈLES EXPLICATIFS DES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES 295
1.4 RECOMMANDATIONS ERGONOMIQUES ..... **ERREUR! SIGNET NON DÉFINI.**

2. CONCLUSION GÉNÉRALE 313

BIBLIOGRAPHIE 321

ANNEXES 335

SOMMAIRE

INTRODUCTION 1

CHAPITRE I
Le cadre de recherche : Aspects généraux de la conception Informatique et des transferts technologiques

1. INTRODUCTION 8

2. LA TÂCHE DE CONCEPTION INFORMATIQUE : DE QUOI PARLONS-NOUS ? 9

3. UNE APPROCHE HISTORIQUE DE LA CONCEPTION INFORMATIQUE 11

3.1 UNE ÉVOLUTION RAPIDE ET MOUVEMENTÉE 11
3.2 EN RÉSUMÉ 16

4. LES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE .. 17

4.1 TRANSFERT TECHNOLOGIQUE : DE QUOI S'AGIT-IL ? 17
4.2 LES IMPLICATIONS HUMAINES DES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE 19
4.3 LES IMPLICATIONS SOCIO-ORGANISATIONNELLES DES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE 21
4.4 EN RÉSUMÉ 22

5. ÉVOLUTION DES OUTILS ET DES MÉTHODES DE CONCEPTION INFORMATIQUE 23

5.1 LES DISPOSITIFS TECHNIQUES DE CONCEPTION INFORMATIQUE 23
5.1.1 Les caractéristiques architecturales des techniques de conception *23*
5.1.2 Les caractéristiques graphiques des techniques de conception *26*
5.1.3 Les caractéristiques logiques des techniques de conception *30*
5.1.4 Les caractéristiques paradigmatiques des techniques de conception *33*
5.1.5 En résumé *37*
5.2 LES DÉMARCHES DE CONDUITE DE PROJET EN CONCEPTION INFORMATIQUE 39
5.1.6 Le cycle de vie du logiciel *39*
5.1.6 Le cycle de vie traditionnel en "V" *40*
5.1.6 Le cycle de vie itératif par Prototypage *46*
5.1.6 En résumé *49*

6. CONCLUSION 50

Chapitre II
Les aspects cognitifs de l'activité de conception informatique

1. INTRODUCTION **52**

2. LE POINT DE VUE DE L'ERGONOMIE DE LA PROGRAMMATION **53**

2.1 ERGONOMIE DES SYSTÈMES TECHNIQUES ET DE L'INFORMATIQUE 53
2.2 GENÈSE DE L'ERGONOMIE DE LA PROGRAMMATION 54
2.3 UN APERÇU DES DOMAINES D'INTERVENTION EN ERGONOMIE DE LA PROGRAMMATION 56
2.4 EN RÉSUMÉ 60

3. LES CARACTÉRISTIQUES COGNITIVES DE L'ACTIVITÉ DE CONCEPTION **61**

3.1 UNE ACTIVITÉ DE RÉSOLUTION DE PROBLÈME 61
3.1.1 Spécificités des problèmes de conception *62*
3.1.2 La conception : entre créativité et routine *63*
3.2 EN RÉSUMÉ 65

4. LES ÉTAPES COGNITIVES DE L'ACTIVITÉ DE CONCEPTION **66**

4.1 LA COMPRÉHENSION DU PROBLÈME 66
4.2 LA CONSTRUCTION D'UNE SOLUTION 68
4.2.1 L'identification et la gestion des contraintes *68*
4.2.2 La mise en oeuvre de connaissances pour l'action : les schémas de connaissances *69*
4.2.3 La définition et l'utilisation de stratégies de conception. *73*
4.3 LA RÉUTILISATION DE SOLUTIONS 77
4.4 LES MODES DE PLANIFICATION 81
4.4.1 Une planification hiérarchisée *81*
4.4.2 Une planification opportuniste *83*
4.4.3 Les facteurs responsables du mode d'organisation de l'activité *85*
4.5 L'IMPLÉMENTATION DE LA SOLUTION RETENUE 86
4.6 EN RÉSUMÉ 91

5. LES TRANFERTS DE COMPÉTENCES DANS LA CONCEPTION **93**

5.1 LE RAISONNEMENT ANALOGIQUE : DE QUOI S'AGIT-IL ? 93
5.2 LES DIFFÉRENTES FONCTIONS DU RAISONNEMENT ANALOGIQUE DANS LA CONCEPTION 94
5.2.1 La compréhension *94*
5.2.2 La résolution *94*
5.2.3 L'apprentissage *95*
5.2.4 En résumé *96*
5.3 LES PRINCIPES DE FONCTIONNEMENT DU RAISONNEMENT ANALOGIQUE 97
5.3.1 Trois modèles explicatifs *97*
5.3.2 Analyse comparative de ces trois modèles *99*
5.3.3 Les limites du raisonnement analogique dans l'activité de conception *101*
5.3.4 Les transferts intra-domaine et les transferts inter-domaine *106*
5.3.5 En résumé *107*
5.4 CARACTÉRISTIQUES DES COMPÉTENCES TRANSFÉRÉES 108
5.4.1 Qu'est-ce que la compétence en conception informatique ? *108*
5.4.2 Les différentes types de compétences transférables *111*
5.4.3 En résumé *119*

6. CONCLUSION **120**

CHAPITRE III
Le travail de recherche sur le terrain

1. CADRE MÉTHODOLOGIQUE 124

1.1 PROBLÉMATIQUE ET HYPOTHÈSES 125
1.1.1 Problème posé 125
1.1.2 Présentation du modèle d'analyse psycho-cognitif 130
1.1.3 Hypothèses 133
1.2 DÉMARCHE GÉNÉRALE D'INTERVENTION 136
1.2.1 La situation du service où s'est déroulée l'étude 136
1.2.2 L'étape de maquettage comme objet d'étude 139
1.2.3 Formalisation du cadre d'observation 140
1.2.4 Démarche méthodologique 141

2. LA DESCRIPTION DU SYSTÈME HOMME-MACHINE : LE CONTEXTE DE L'ANALYSE 146

2.1 LES CARACTÉRISTIQUES TECHNIQUES DES ENVIRONNEMENTS DE CONCEPTION ÉTUDIÉS 147
2.1.1 Méthode employée 147
2.1.2 Les caractéristiques architecturales des environnements de conception étudiés 148
2.1.2.1 L'architecture centralisée du site-central 148
2.1.2.2 L'architecture distribuée du client serveur 149
2.1.2.3 Synthèse comparative 149
2.1.3 Les caractéristiques fonctionnelles des environnements de conception étudiés 150
2.1.3.1 Spécificités fonctionnelles de Pacbase 150
2.1.3.2 Spécificités fonctionnelles de NSDK 154
2.1.3.3 Synthèse comparative 155
2.1.4 Les caractéristiques graphiques des environnements de conception étudiés 157
2.1.4.1 L'interface alphanumérique de Pacbase 157
2.1.4.2 L'interface graphique de NSDK 159
2.1.4.3 Syntèse comparative 163
2.1.5 Les caractéristiques logiques des environnement de conception étudiés 165
2.1.5.1 Un paradigme de conception transactionnel et synchrone sur le site-central 165
2.1.5.2 Un paradigme de conception événementiel et asynchrone sur client serveur 165
2.1.5.3 Synthèse comparative 169
2.1.6 Synthèse - Discussion 170

2.2 L'ORGANISATION DE L'ACTIVITÉ DE MAQUETTAGE 174
2.2.1 Méthode employée 174
2.2.2 Organisation de l'activité de maquettage avec Pacbase 175
2.2.3 Organisation de l'activité de maquettage avec NSDK 179
2.2.4 Synthèse - Discussion 185

2.3 L'ANALYSE DES CONTRAINTES ET DES ASTREINTES 190
2.3.1 Méthode employée 190
2.3.2 Analyse des contraintes 192
2.3.3 Analyse des astreintes 200
2.3.4 Synthèse - Discussion 201

2.4 CONCLUSION GÉNÉRALE SUR L'ANALYSE DU SYSTÈME HOMME-MACHINE 204

3. L'ANALYSE COGNITIVE DE L'ACTIVITÉ DE MAQUETTAGE 207

3.1 LE MODÈLE D'ACTION 208
3.1.1 Méthode employée 208
3.1.2 Présentation des résultats 211
3.1.2.1 La planification dans Pacbase 213
3.1.2.1.1 Structure de buts définie pour la tâche de maquettage sur Pacbase 213
3.1.2.1.2 Une planification hiérarchisée de l'activité chez les experts Pacbase 215
3.1.2.2 La planification dans NSDK 219
3.1.2.2.1 Structure de buts définie pour la tâche de maquettage dans NSDK 219
3.1.2.2.2 Une planification située chez les experts NSDK 222
3.1.2.2.3 Une planification hiérarchisée puis opportuniste de l'activité chez les novices NSDK 223
3.1.2.2.4 En résumé 226
3.1.2.3 Les capacités à changer de plan 228
3.1.3 Synthèse - Discussion 230

3.2 MODÈLE DE RÉSOLUTION 233
3.2.1 Méthode employée 233
3.2.2 Aspects quantitatifs des solutions déployées durant l'activité de maquettage 237
3.2.2.1 Présentation des résultats 237
3.2.2.2 Analyse des résultats 238
3.2.3 Aspects qualitatifs des solutions déployées durant l'activité de maquettage 247
3.2.3.1 Présentation des résultats 247
3.2.3.2 Analyse des résultats 248
3.2.3.2.1 Nature des problèmes rencontrés 248
3.2.3.2.2 La représentation du problème 250
3.2.3.2.3 Processus de résolution 251
3.2.3.2.4 Traitements mis en œuvre 259
3.2.3.2.5 Processus d'évaluation et d'ajustement des solutions 262
3.2.4 Synthèse - Discussion 265

3.3 MODÈLE D'INTERACTION 271
3.3.1 Méthode employée 271
3.3.2 Présentation des résultats 273
3.3.3 Analyse des résultats 274
3.3.3.1 L'agencement des écrans sur Pacbase : une construction par codage 274
3.3.3.2 L'agencement des fenêtres avec NSDK : une conception par manipulation directe 277
3.3.4 Synthèse-Discussion 281

3.4 MODÈLE DE RÉFÉRENCE 283
3.4.1 Méthode employée 283
3.4.2 Présentation des résultats 284
3.4.3 Analyse des Résultats 285
3.4.3.1 Description des connaissances utilisées dans chaque environnement 286
3.4.3.2 Modèle de référence utilisé sur Pacbase : stabilité des références 291
3.4.3.3 Modèle de référence utilisé sur NSDK : instabilité des références 291
3.4.4 Synthèse - Discussion 293

CHAPITRE IV
Discussion générale et Conclusion

1. DISCUSSION GÉNÉRALE 295

1.1 LES MODÈLES MENTAUX EN PRÉSENCE 295
1.2 PRINCIPALES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES 297
1.3 DEUX MODÈLES EXPLICATIFS DES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES 295
1.3.1 Du rôle des mécanismes de transfert d'apprentissage dans la réutilisation de compétences erronées 299
1.3.2 Du rôle formateur de l'environnement sur les conduites cognitives de l'informaticien..... 304
1.4 RECOMMANDATIONS ERGONOMIQUES 305

2. CONCLUSION GÉNÉRALE 313

BIBLIOGRAPHIE 321

ANNEXES

Annexe 1 : Principes et modalités d'intervention de la Recherche-Action 336

Annexe 2 : Situation de l'entreprise où s'est déroulée l'étude 338

Annexe 3 : Les méthodes de recueil de données et les problèmes posés par les techniques de verbalisation 340

Annexe 4 : Grille d'évaluation ergonomique de Pacbase et NSDK 343

Annexe 5 : Structure logique d'un squelette de programmation TP 354

Annexe 6 : Justification de la référence au modèle GOMS 355

Annexe 7 : Exemple de représentation papier de la maquette à réaliser par les informaticiens novices sur NSDK 356

Annexe 8 : Lexique technique et fonctionnel du maquettage 357

"Être compétent,
c'est se tromper selon les règles"
- Paul VALERY

"Mes clients sont libres de choisir
la couleur de leur voiture,
à condition qu'ils la veuillent noir"
- Henri FORD

"L'expérience est le nom que
chacun donne à ses erreurs"
- Oscar WIlDE

INTRODUCTION

A l'aube du XXI[e] siècle, l'informatique professionnelle est confrontée à trois grandes échéances majeures qui l'obligent à revoir ses méthodes et ses techniques de travail : il s'agit du passage à l'an 2000, de la conversion des systèmes monétaires à l'Euro et de la généralisation de l'informatique de réseau (client serveur, intranet/internet) et orientée objet. Si les deux premières nécessitent une main d'œuvre plus importante (plus de 27 000 informaticiens selon les estimations de l'APEC[1]), les mutations techniques requièrent, en revanche, du personnel plus qualifié et sachant s'adapter très rapidement à de nouvelles manières de penser et d'organiser la programmation. Une prévision[2] de la Délégation à la Formation Professionnelle relevait d'ailleurs que l'arrivée de ces nouvelles technologies allait entraîner la reprofessionnalisation de près de 37 000 des 312 000 informaticiens français.

Plus généralement, changer de technique de conception revient à introduire de nouveaux principes de travail dans un ensemble de règles socioprofessionnelles dont l'équilibre trouve ses racines dans un réseau d'habitudes. De fait, cet équilibre est nécessairement rompu. Les informaticiens sont alors amenés à apprendre ou à réapprendre des comportements de travail, des logiques d'activité, des stratégies cognitives qui répondent aux exigences de ces nouveaux dispositifs. Ces changements correspondent, en somme, à des phases de rupture des acquis professionnels, et soulèvent des questions plus générales concernant l'adaptation et la reconversion des informaticiens en place.

De ce point de vue, la réussite d'un projet de migration technologique dépend de la manière dont le système humain réagit aux différentes évolutions, qu'elles soient techniques, fonctionnelles ou organisationnelles. Autrement dit, ce qui détermine le succès d'un tel programme, c'est moins la complexité des techniques elles-mêmes que la gestion cohérente d'une série de ruptures relatives aux modalités de raisonnement, à la transmission des savoir-faire, et aux conditions d'organisation de l'activité. Dès lors, aussi complexes que soient ces techniques, aussi profondes que soient les transformations qu'elles impliquent ; les enjeux de l'ère de l'informatisation appellent les entreprises à anticiper et à gérer au mieux les effets du changement.

---

[1] Agence Pour l'Emploi des Cadres

[2] "*Le Monde Informatique*" daté du 4 février 1994

Notre recherche, réalisée dans le cadre d'une convention CIFRE[3], s'inscrit dans cette problématique. Elle répond à une demande d'une entreprise du tertiaire, spécialisée dans le domaine de la prestation informatique, et qui se trouve aux prises directes avec l'évolution de ses environnements de conception : 300 développeurs doivent ainsi basculer d'un système de conception procédural sur une architecture centralisée (site-central) vers un système orienté objet et une architecture distribuée (client serveur). L'objectif de notre intervention est de sérier les obstacles et les difficultés que cette migration entraîne, et de réfléchir quant aux moyens à mettre en œuvre pour accompagner la reconversion du personnel.

Cette intervention, en psychologie ergonomique de la programmation, s'inscrit plus généralement dans le courant de la "recherche-action[4]" (Dubost, 1987). Elle prend la forme d'une collaboration entre un chercheur et une organisation pour produire, sur la base de diagnostics et des théories préétablies, un nouveau processus corrigeant et/ou optimisant une situation existante.
Cela dit, aborder l'intervention sous cet angle pose quatre problèmes qui tiennent respectivement :

1. aux enjeux politiques et sociaux du phénomène étudié ;
2. à la pluralité des points de vue et des perspectives adoptés pour le décrire ;
3. au respect de l'existant ;
4. à l'absence de "contrôle" de la situation analysée.

Il est intéressant d'aborder rapidement ces différents points car ils conditionnent les termes et les conditions mêmes de l'intervention.

[3] Convention Industrielle de Formation et de Recherche en Entreprise passée entre le ministère de la Recherche (ANRT), une entreprise du secteur privé et un laboratoire de recherche universitaire.
[4] Elle repose sur un certain nombre de principes et de modalités d'intervention qui sont présentés dans l'Annexe I de cette thèse.

Les contraintes inhérentes à la recherche appliquée

1. *Les enjeux politiques et sociaux du phénomène étudié*

   Tout changement, qu'il soit technique ou organisationnel, suscite des incertitudes et des craintes qui affectent le climat social de l'entreprise. Il en est ainsi du service au sein duquel s'est déroulée notre intervention. Les informaticiens s'interrogeaient ainsi beaucoup sur leur avenir professionnel, redoutant d'être mis à l'écart par manque de compétences techniques. Dans ce contexte sensible, les responsables craignaient que notre intervention ne ravivent et/ou n'accentuent encore davantage les inquiétudes du personnel. Notre marge de manoeuvre en fut d'autant plus réduite : toute réunion avec les informaticiens devait être préalablement planifiée et préparée avec la hiérarchie. De même, le recueil de données sur le terrain était souvent précédé d'une longue période de négociation où il fallait justifier l'intérêt de telles méthodes. Enfin, plusieurs techniques d'investigation ont été rejetées parce qu'elles paraissaient trop tendancieuses aux yeux de la direction. Ce fut notamment le cas de l'expérimentation en milieu naturel, des questionnaires, des enregistrements vidéo.

2. *Pluralité des points de vue et des perspectives adoptés pour décrire le phénomène*

   Un autre problème découlant de la recherche appliquée concerne le rôle de l'intervenant dans l'entreprise : doit-il rester un chercheur focaliser sur son objet scientifique, peut-il endosser la veste de consultant comme l'entreprise l'y pousse, a-t-il les moyens et les capacités de se prévaloir de ces deux titres à la fois ? La réponse est loin d'être triviale.

   Le chercheur doit s'efforcer d'harmoniser des attentes et des contraintes émanantes des différents partenaires (universitaire et entreprise), qui se révèlent pourtant souvent divergentes, voire contradictoires. En effet, d'un côté, l'entreprise réclame une analyse rapide et concrète, et juge l'efficacité de l'intervention en fonction des résultats obtenus (gain de productivité, amélioration des processus...). De l'autre, le travail de recherche exige une démarche méthodologique rigoureuse pour étudier et expliquer les phénomènes en jeu.

   Pour satisfaire ces attentes, l'option choisie a été de présenter de façon discursive, linéaire, une analyse qui rend compte à la fois des aspects diachroniques (le récit

des actes et des événements) et synchroniques (les éléments structuraux en rapport avec le processus) du phénomène étudié.

3. *Respect de l'existant*

Un autre exigence de ce type de recherche est de tenir compte des contraintes professionnelles ; autrement dit, assurer aux personnes que l'intervention ne grèvera ni leur niveau de productivité, ni la qualité de leur travail. Il y a donc des choix à opérer, des compromis à trouver sur la démarche à mettre en œuvre et les techniques à utiliser. Ainsi, tout ce qui était susceptible d'accentuer leur charge de travail –comme par exemple, les tests ou les expérimentations (test de la double tâche pour évaluer les astreintes)– a dû être abandonné au profit d'autres techniques moins lourdes. Pour compenser ce manque de disponibilité, un plus grand nombre de personnes a été impliqué dans la recherche (près d'une trentaine).

4. *Absence de contrôle de la situation analysée*

Un reproche souvent fait à la recherche en milieu naturel est qu'elle ne tient pas compte des nombreuses variables qui peuvent affecter à la fois la situation d'observation (par exemple, les pressions hiérarchiques ou syndicales) et les phénomènes observés (variables parasites, variables secondaires non contrôlées). Nous pensons au contraire que c'est justement ce qui fait tout l'intérêt et toute la richesse de ce type d'intervention. Les processus étudiés sont en effet produits dans et pour un contexte de travail spécifique et original, que l'expérimentation n'est pas en mesure de restituer. En contrôlant un trop grand nombre de variables, l'expérimentation finit par créer une situation dénaturée, voire artificielle, qui est très éloignée des réalités du terrain. Se pose dès lors la question de la généralisation de ses résultats expérimentaux.

Organisation de la recherche

Le plan de cette recherche s'organise de la façon suivante :

1) Dans un **premier chapitre**, nous situerons le *cadre général* de cette recherche en définissant successivement la conception informatique, les évolutions qu'elle a subies depuis les quarante dernières années, les transferts technologiques et leurs impacts dans le secteur informatique (en tant qu'ils requièrent des transferts d'apprentissage et renvoient à un déterminisme technologique) et enfin, les principales innovations techniques et méthodologiques qui ont transformé l'activité de conception.

2) Le **second chapitre**, sur les aspects fondamentaux et théoriques, telles les spécificités de l'activité de conception, et les processus cognitifs impliqués dans la résolution informatique et dans les transferts de compétences, présente les différents modèles et les principaux concepts sur lesquels nos travaux sont basés. Nous abordons également l'ergonomie de la programmation en tant qu'elle fournit, par ses études expérimentales et empiriques, des théories sur le fonctionnement humain dans le cadre de la conception informatique

3) Le **troisième chapitre** expose notre démarche d'intervention et présente les résultats obtenus par l'enquête de terrain. L'organisation de cette partie se déroule en trois temps :
   - D'abord, nous commencerons par préciser le cadre méthodologique de notre intervention, en définissant la problématique et les hypothèses de recherche, le contexte de l'intervention et la méthode employée pour recueillir les données, *via* un modèle d'analyse psycho-cognitif.
   - Ensuite, nous décrirons le système homme-machine relatif à chaque contexte de conception : les aspects techniques et organisationnelles de chaque environnement de programmation source et cible, ainsi que les contraintes et les astreintes qu'ils génèrent sur les informaticiens.
   - Enfin, l'examen cognitif des conduites de conception propres à chaque environnement technique et à chaque groupe d'informaticiens (experts

Pacbase, experts NSDK et novices NSDK) sera effectué. Pour ce faire, nous utiliserons notre modèle d'analyse qui permettra de dégager quatre types de compétences en œuvre dans : *(i)* l'organisation de l'activité de maquettage (*modèle d'action*), *(ii)* la résolution des problèmes (*modèle de résolution*), *(iii)* la coopération homme-machine (*modèle d'interaction*) et *(iv)* la référence à des modèles de conception (*modèle de référence*). Nous indiquerons également les difficultés que rencontrent les informaticiens novices lors de l'appropriation du nouveau dispositif.

4) Le **quatrième et dernier chapitre** comporte la discussion générale et la conclusion.

   La discussion sera l'occasion de confronter nos résultats aux hypothèses. Nous réfléchirons aussi sur les raisons qui conduisent les informaticiens novices à effectuer de mauvais transferts d'apprentissage. Un modèle explicatif de ces dysfonctionnements cognitifs sera d'ailleurs proposé. Une série de recommandations ergonomiques, développées à partir de nos analyses et observations, viendra clore cette discussion. Elles ont pour but d'accompagnement le changement technologique.

   La conclusion, quant à elle, établira le bilan général de cette recherche et présentera différentes perspectives sur lesquelles pourraient déboucher des recherches concernant l'étude des transferts d'apprentissage en conception informatique

CHAPITRE 1

Le cadre de recherche : Aspects généraux de la conception informatique et des transferts technologiques

Résumé

L'objectif de ce chapitre est de préciser le cadre de notre recherche, à travers une approche de la conception informatique et des transferts technologiques qui l'affectent.

Pour ce faire, nous définissons la tâche de conception informatique comme un processus de développement global qui organise, définit, répartit et coordonne l'activité de différents acteurs, afin de transformer progressivement un besoin initial (demande du client) en un résultat final (un dispositif symbolique).

Nous montrons aussi que ce secteur professionnel a connu l'une des évolutions technologiques et fonctionnelles les plus mouvementées de l'après-guerre. Cette progression s'effectue au rythme des nombreuses innovations techniques qui se succèdent depuis les années quarante, ainsi que des politiques d'informatisation massives qui sont mises en œuvre par les entreprises.

Nous discutons également des transferts technologiques, et plus précisément, de ses implications humaines et organisationnelles sur la conception informatique. On relève que deux types de phénomène sont inhérents à ces changements : (i) les transferts d'apprentissage, (ii) le déterminisme technologique.

Pour illustrer ces propos, nous montrons comment l'évolution des outils et des méthodes de conception informatique transforme les manières de penser et d'organiser le développement.

1. INTRODUCTION

La conception informatique est un cas particulier de la conception. Rapidement, il s'agit de programmer des dispositifs virtuels à l'aide de systèmes techniques perfectionnés, en suivant des démarches de conception particulières. Des premiers ordinateurs site central aux derniers systèmes client serveur, des cartes perforées antédiluviennes aux récents langages de programmation orientés objets, la conception a été traversée par de très nombreuses innovations techniques qui l'ont conduite à revoir ses modes d'intervention et à élargir ses champs de compétences. La conception informatique apparaît comme une situation hautement instable parce qu'extrêmement sensible et réactive à l'environnement technique dans lequel elle évolue. On parlera d'ailleurs de transfert technologique pour désigner l'irruption, toujours plus marquée, des nouvelles technologies dans ce domaine d'activité. Cela dit, ces changements ne symbolisent pas uniquement le passage d'un système obsolète vers un système plus évolué. Toute une série d'effets secondaires accompagnent ce processus de modernisation. C'est à l'identification de ces effets qu'est en partie consacrée ce chapitre.

Pour ce faire, après avoir précisé ce qu'est la tâche de conception dans le domaine informatique, nous retraçons son évolution historique depuis les années quarante. Ensuite, nous discuterons des transferts technologiques et de leurs conséquences sur la conception informatique. Enfin, nous présenterons des illustrations plus concrètes de ces effets, en montrant dans quelle mesure l'évolution des techniques de conception (outils et démarches) peut transformer les pratiques de l'informaticien.

2. LA TÂCHE DE CONCEPTION INFORMATIQUE : DE QUOI PARLONS-NOUS ?

De nombreux domaines professionnels requièrent la mise en œuvre d'activités de conception : architecture, ingénierie mécanique, aérospatiale, etc. Mais dans le domaine de l'informatique, la conception informatique peut être évoquée de deux manières :

a) La première consiste à l'appréhender comme une démarche structurée alternant différentes phases *–analyse, spécification fonctionnelle, développement, tests et maintenance–* dont la finalité est la génération de dispositifs virtuels et symboliques (applications informatiques, programmes, interfaces graphiques) à partir de besoins exprimés par des utilisateurs (Winograd, 1995 , Darses et Falzon, 1996).

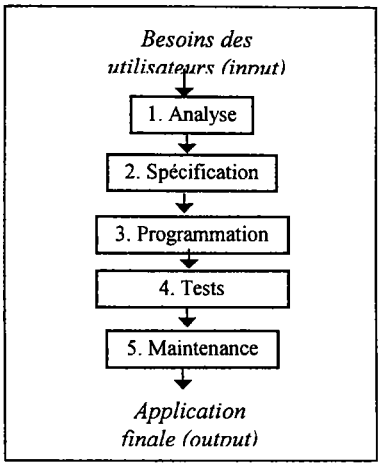


Figure 1
La conception informatique vue comme un processus de développement menant d'un état initial vers un état final

b) La seconde, empruntée aux domaines de l'ingénierie et de l'organisation industrielle, est une approche plus fonctionnelle. Elle oppose la *conception* à la *réalisation* : d'un côté, il y ceux qui pensent et qui décident (*les concepteurs ou analystes*), et de l'autre, ceux qui exécutent et suivent les prescriptions issues de la conception (*les réalisateurs ou exécutants*) (Hatchuel, 1996). Rapportée à l'informatique, la conception correspond donc aux phases situées en amont de la démarche (*"Analyse" et "Spécification"*) ; la réalisation, aux phases en aval (*"Programmation et Maintenance"*). De la sorte, ceux qui pensent et décident (les analystes) prescrivent le travail à ceux qui réalisent (les programmeurs). Et les décisions des uns structurent l'activité des autres.

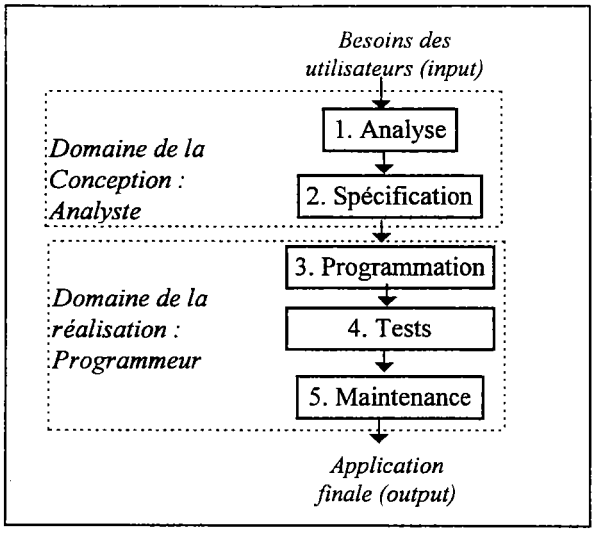


Figure 2
Approche fonctionnelle de la conception informatique :
La conception est une étape particulière du processus de développement.

En ce qui nous concerne, et d'après les nouvelles méthodes de conception[5] itératives qui tendent à bannir les frontières entre "analyse" et "programmation", nous considérons la conception informatique comme un processus de développement intégré qui organise, définit, répartit et coordonne l'activité de tous les acteurs, afin de transformer progressivement un besoin initial (demande du client) en un état final (un dispositif symbolique). C'est pourquoi, nous utiliserons d'ailleurs indifféremment les termes de programmation, de développement ou de conception pour désigner cette activité. De même, concepteur, développeur et programmeur renverront à la fonction générale d'informaticien exerçant cette tâche de conception. Enfin, on parlera aussi d'utilisateur ou d'opérateur pour qualifier les informaticiens utilisant le système informatique.

La partie suivante retrace l'évolution de la conception informatique depuis les cinquante dernières années. On verra ainsi comment cette activité a été amenée à redéfinir et/ou à élargir ces domaines de compétences en fonction des découvertes techniques et des stratégies des entreprises.

[5] Méthodes par prototypage et "RAD" (Rapid Application Devlopment).

3. UNE APPROCHE HISTORIQUE DE LA CONCEPTION INFORMATIQUE

L'activité de conception informatique dans les entreprises connaît aujourd'hui, avec Internet et les technologies orientées objet, un essor important. Néanmoins, son développement est le fruit d'une longue histoire, débutée depuis les années 40 et marquée par des ruptures et des évolutions spectaculaires. Partant de la rétrospective proposée par Delmond (1995), et l'enrichissant par les travaux de Breton (1987); Fischer, (1991) ; Desaintquentin et Sauteur, (1991) ; Birrien (1992) ; Buckland et Liu, 1995 ; Ducateau (1995) ; Galland (1995) et Sybord (1995) ; l'évolution de l'activité informatique peut se découper en cinq grandes périodes.

3.1 UNE ÉVOLUTION RAPIDE ET MOUVEMENTÉE

a) L'après-guerre ou l'aventure scientifique et les "mathématiciens-programmeurs"

Sans entrer dans les méandres de l'Histoire qui font remonter l'invention de l'informatique (vue comme un traitement rationnel de l'information) à l'époque des Sumériens, la naissance de l'informatique peut être datée au lendemain de la Seconde Guerre mondiale, en 1946 plus exactement, avec le rapport de von Neuman (Birrien, 1992). Ce professeur en mathématique de Princeton y présentait les concepts fondateurs de l'ordinateur selon l'architecture dite "von Neuman". Faisant table rase des contingence matérielles, il fournit la première définition scientifique du fonctionnement de la machine. C'est dans ce contexte que fut inventée l'ENIAC[6], premier ordinateur binaire de la génération, capable de traiter près d'un million de cartes perforées à son premier essai.

Durant cette période, qui va s'étendre jusqu'aux années 60, l'informaticien se trouve confronté à une technique encore expérimentale et il présente de nombreuses caractéristiques communes avec l'archétype du chercheur : connaissances approfondies, apprentissage constant, mais aussi isolement dans la tâche (Fischer, 1991).

b) Les années 70 et l'institution d'une classe d'experts

Les années 70 marquent la période de diffusion de l'informatique. Pour assurer et soutenir la propagation de cette technologie, les entreprises vont créer des

[6] ENIAC : Electronic Numerical Integrator Computer

directions informatiques qui auront la responsabilité de la ressource informatique. Le discours technique y domine sans partage. Employant un langage ésotérique et abscons pour les profanes, les informaticiens sont perçus alors comme des personnes inaccessibles et difficilement contrôlables. La complexité des systèmes informatiques représente l'instrument de légitimation de leur pouvoir dans l'entreprise. Nul ne peut contester l'autorité de l'informaticien s'il n'est pas reconnu comme étant lui-même un expert du domaine.

Tout cela participe finalement à la constitution d'une caste d'experts au sein même de l'entreprise : ils se reconnaissent entre eux par des signes distinctifs (langages techniques, formations communes...), communiquent peu avec les autres et ont souvent des objectifs différents de ceux de l'entreprise (davantage axés sur les recherches et les découvertes que sur la productivité et la rentabilité).

C'est aussi la période des directions informatiques toutes puissantes, où des sommes considérables sont engagées pour le processus d'informatisation des situations de travail. Les informaticiens ont alors la responsabilité des orientations techniques. Fort de cette position, l'informatique devient l'une des composantes stratégiques de l'essor des entreprises.

Cependant, malgré cette domination, cette informatique reste encore très "artisanale", car ses méthodes et ses techniques sont encore peu formalisées et relèvent d'un savoir-faire particulier.

c) L'ébranlement des années 80

Les innovations technologiques majeures que connaît l'informatique (par la miniaturisation des composants électroniques notamment) commencent à se diffuser dans les entreprises. C'est l'ère encore "balbutiante" de la micro-informatique et des systèmes d'exploitation ouverts, de type Unix et Window. Le dialogue homme/machine fait des progrès considérables, grâce aux découvertes chercheurs du Xerox-PARC[7]. L'image de l'informatique renoue ainsi avec l'innovation et la créativité des premiers chercheurs, mais subit un glissement significatif : ce sont dorénavant des "petits génies du bricolage" qui, par la recherche de nouvelles sources de convivialité, sont à l'origine des plus grandes

[7] Pala Alto Research Center

découvertes du domaine (Steve Job pour l'interface conviviale du Mac, Bill Gates pour le système d'exploitation Windows...).

Au sein des entreprises, les directions informatiques et les informaticiens tiennent bon. Elles affirment d'autant plus leur suprématie qu'une campagne de recrutement sans précédent va être lancée. Les services informatiques manquent en effet cruellement de ressources face à la demande, toujours plus pressante, des secteurs opérationnels. Pour y remédier, les entreprises doivent se résoudre à embaucher du personnel non qualifié techniquement (géologie, biologie, mathématique, agronomie....) qu'elles se chargeront de former en interne.

Les missions qui leur sont confiées sont relativement spécialisées et tournent autour de quatre pôles d'activité :

1. Dans le premier pôle, on trouve des analystes chargés de faire des spécifications détaillées et fonctionnelles sur le domaine à informatiser.
2. Dans le second, ce sont des développeurs qui ont pour mission de coder et d'implémenter les spécifications du groupe d'analystes dans le système informatique.
3. Le troisième pôle, chargé de la maintenance, a pour but d'intervenir sur les applications en cas de dysfonctionnement ou d'évolution des versions, et d'assister les utilisateurs lorsqu'ils rencontrent des problèmes.
4. Le dernier, enfin, réunit des personnes chargées de la gestion des systèmes de base de données (SGBD) du site-central.

L'activité informatique tend alors à se rationaliser. Le découpage des tâches marque ainsi la spécialisation des compétences et la parcellisation de l'activité. On est ainsi dans le cas de la démarche structurée décrite dans la première partie, séparant les analystes des programmeurs, les concepteurs des réalisateurs.

d) Début des années 90 : La rupture

Le début des années 90 s'illustre par une fracture de l'ordre établi : les référents théoriques (architecture centralisée), culturels (respect de l'expertise scientifique et technique des informaticiens) et organisationnels (le pouvoir des directions informatiques) sont ébranlés par l'avènement des systèmes décentralisés dans l'entreprise (client serveur et réseau). En outre, l'informatique devient une

ressource banalisée grâce à la diminution de ses coûts de fabrication et à une ergonomie très conviviale.

On assiste alors à une véritable désacralisation de la fonction informatique. Sur le plan financier, les décisions d'investissements techniques sont désormais prises en concertation avec les utilisateurs. De même, l'informaticien coopère étroitement avec l'utilisateur final dans la conception du projet, et ne cherche plus à imposer ses points de vue. En effet, se référant aux avantages et à la convivialité de la micro-informatique, les utilisateurs ne sont plus dupes des propositions de l'informatique, et deviennent extrêmement exigeants quant à la qualité, aux coûts et aux délais des prestations fournies. La balance des pouvoirs s'inverse donc au profit des opérateurs qui acquièrent une culture technique variée et multiforme, qui ne cesse de se développer depuis (Cohen, 90).

Pour collaborer davantage avec ses clients, les informaticiens sont obligés de faire évoluer leur méthode d'intervention, qui était jusqu'alors trop technique, vers une approche plus gestionnaire et sociale. Ce changement implique qu'ils s'ouvrent aux "affaires humaines", c'est-à-dire qu'ils fassent preuve de qualité d'écoute et de pragmatisme pour l'accompagnement des utilisateurs durant tout le processus d'informatisation de leur poste de travail (Frechin, 1994). Mais cela nécessite aussi que l'informaticien acquière les compétences idoines, notamment relationnelles et de négociation.

Enfin, pour diminuer les coûts de l'informatique, les entreprises tendent à externaliser la conception vers des sociétés extérieures (SSII)[8] (Pluchart, 1997). Face à ces nouveaux concurrents, la position de l'informaticien dans l'entreprise se fragilise encore un peu plus.

Finalement, l'informaticien des années 90 est un développeur enfin accessible, et proche des utilisateurs. Intervenant à toutes les étapes du cycle de développement (analyse détaillée, conception d'interfaces, réalisation des programmes et maintenance de ceux-ci), c'est aussi un informaticien polyvalent qui maîtrise l'ensemble du processus de conception (Sybord, 1995).

[8] SSII : Société de Services et d'Ingénierie en Informatique

e) Fin des années 90 : un tendance à l'industrialisation de l'informatique

En cette fin de siècle, deux événements majeurs marquent le grand retour des informaticiens dans l'entreprise : D'abord, le passage des logiciels à l'an 2000 et à l'Euro conduit au rappel de toute une catégorie d'informaticiens spécialisés sur les anciens systèmes centralisés. Paradoxalement, ce sont ceux-là même que les entreprises n'avaient pas hésiter à licencier au début des années 90, en raison de leur inadaptation supposée aux nouvelles architectures client serveur. Ensuite, l'essor considérable des technologies Internet/Intranet nécessite le recrutement massif d'informaticiens spécialisés sur de nouveaux domaines : ingénieurs réseaux et architectures, graphistes, développeurs HTML et Java.... Au total, la pénurie d'informaticiens se chiffrerait[9] à plus de 27 000 personnes en France et à près de 300 000 pour les Etats-Unis.

Mais dans le même temps, la recherche d'une plus grande rationalisation du processus de développement devient le souci dominant des organisations. Elles se tournent alors vers des langage orientés objets qui permettent, grâce au principe de réutilisabilité, des gains de productivité de l'ordre 7 à 10% dans les cas les moins favorables (Coulange, 1993). D'aucuns vont même jusqu'à pronostiquer une nouvelle transformation du métier d'informaticien ; avec d'un côté, ceux qui créent et conçoivent ces composants (à l'extérieur de l'entreprise), et de l'autre, ceux qui les assemblent (à l'intérieur de l'entreprise) pour former l'application finale (Kakagiakos, 1993).

En somme, après avoir connu une période d'intrication étroite entre ses différentes fonctions, l'informatique de demain tend vers une nouvelle rationalisation de son activité en séparant la fonction de "*concepteur de composants logiciels*" de celle "*d'assembleur*".

[9] Source : APEC

3.2 EN RÉSUMÉ

Finalement, au cours de ces cinquante dernières années, les pratiques de conception ont connu des mutations très rapides, survenues à l'occasion de nombreuses découvertes techniques (gros système, micro-informatique, architecture client serveur, internet...). Les cinq grandes mutations que nous avons mises en évidences (Cf Figure 3 plus bas) ont conduit à l'élargissement des compétences de l'informaticien (techniques, humaines et sociales), au recadrage de sa fonction et de son pouvoir au sein de l'organisation, et enfin à une transformation des modalités d'organisation de son activité. L'informaticien appartient ainsi à un espace professionnel qui ne cesse de bouger et de se remettre en cause. Cette instabilité chronique exige de sa part un repositionnement constant en terme de savoirs (formations à acquérir...), de savoir-faire (compétences et qualifications à développer) et de savoir être (recentrage dans le jeu socio-organisationnel).

	Les pionniers 1950/60	**La classe d'experts Années 70**	**L'ébranlement : Années 80**	**La rupture : Années 90**	**Nouvelle tendance de la fin des années 90**
– **Tendances techniques**	Expérimentation création des outils	Informatique centralisée	Révolution de la micro-informatique	Informatique distribuée	Composants réutilisables et nouveaux langages orientés objets
– **Diffusion de l'informatique**	Quelques très grandes entreprises	Moyennes et grandes entreprises	Généralisation	Réseaux, services aux particuliers	village planétaire (Internet : réseau des réseaux)
– **Image de l'informatique**	Peur d'une technologie dominatrice	Légitimité du management rationnel	Retour à la créativité avec la micro-informatique Rejet de l'informatique classique	"Révolution informationnelle" : les aspects techniques passent au second plan	Retour à la technique, industrialisation, externalisation , délocalisation
– **Image de l'informaticien**	Mathématicien chercheur	Expert membre d'une caste	Petits Génies au fond de leur garage	Désacralisation	Professionnels experts : Ingénieurs
– **Relations informaticiens / utilisateurs**		Direction informatique très puissante	Utilisateur sous la dépendance de l'informaticien	Responsabilités rendues aux utilisateurs	Informaticiens au service, aux ordres et à l'écoute de l'utilisateur
– **Contrôle exercé sur l'informatique**		Quasi inexistant	Refacturation des coûts aux utilisateurs	Limitation "stricte" des budgets informatiques	Notion de productivité et de rentabilité
– **Compétences de l'informaticien**	Recherche	Techniques (monotâche)	Techniques et fonctionnelle (spécialiste ou généraliste)	Techniques, relationnelles, fonctionnelles, ... (Polycompétent)	Retour à des spécialistes techniques et fonctionnels

Figure 3
Les implications des transferts technologiques dans le domaine informatique (D'après Delmond, 1995 : tableau repris et enrichi)

L'évolution de la conception informatique renvoie donc directement aux changements techniques qui la touche. Nous discutons ci-dessous des implications et des enjeux de ces transferts technologiques sur la situation de conception.

4. LES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE

4.1 TRANSFERT TECHNOLOGIQUE : DE QUOI S'AGIT-IL ?

Avant de traiter plus spécifiquement des transferts technologiques, nous allons tout d'abord définir le concept de technologie, car les acceptions particulières auxquelles il renvoie permet de mieux saisir les enjeux véritables des transferts informatiques.

a) La technologie

Il existe deux sens généralement admis pour définir la technologie :

- Le premier, étymologique, la perçoit comme la science des techniques. En clair, la technologie cherche à saisir les pratiques empiriques relatives à la technique et à les normaliser, ainsi qu'à découvrir des perfectionnements. En ce sens, la technologie « *est l'intermédiaire entre la science et la technique* ». (Guegan, Rosanvallon, Troussier, p 34, 1987). Ce n'est pas cette définition qui nous intéresse.
- En revanche, la deuxième, plus classique, considère la technologie comme « *une technique moderne et complexe* » (définition tirée du "*Nouveau Petit Robert*", 1994).

A première vue, cette seconde définition correspond davantage à notre approche. Pourtant, affirmer que la technologie est le synonyme compliqué du mot "*technique*" est un raccourci sémantique quelque peu réducteur. Le risque est en effet d'appréhender le changement informatique comme une simple évolution technologique, c'est-à-dire comme le remplacement d'outils jugés obsolètes par des techniques plus modernes de développement. Nous pensons, bien au contraire, que l'arrivée de ces technologies joue un rôle plus fondamental dans l'activité humaine et qu'elles conditionnent certaines de ses dimensions. D'où cette question qui nous paraît essentielle : quelle est la nature des relations qui existent entre l'homme et la technologie ? Quel rôle joue exactement ces technologies dans l'activité humaine ?

Un premier élément de réponse est apporté par Rabardel (1995) qui considère la technologie comme une entité intermédiaire, un moyen terme, voire un univers intermédiaire entre le sujet (ici, l'informaticien) et "l'objet" sur lequel porte son action (des dispositifs virtuels : programmes, interfaces...). Sa position intermédiaire en fait

un médiateur des relations entre le sujet et l'objet : elle est "*instrument de médiation pragmatique*" lorsqu'elle est le moyen d'une action transformatrice dirigée vers l'objet et un "*instrument de médiation épistémique*" lorsqu'elle permet la connaissance sur l'objet.

Cette approche est séduisante car elle suppose finalement que la technologie est à la fois un moyen d'action et une source de connaissances pour l'individu. Elle lui donne la possibilité d'agir sur son environnement et d'obtenir, en retour, des informations sur le résultat de ses actions. Et c'est à partir de ces informations qu'il pourra réajuster son activité. En somme, les technologies informatiques auraient donc un rôle fondateur dans la structuration des pratiques de travail de l'individu

D'autres éléments de réponse sont également fournis par Dobrov (1979). Il entrevoit la technologie comme une « *mise en oeuvre réfléchie et organisée de la technique* ». La technologie serait porteuse de moyens techniques (*Hardware*), mais aussi de principes et méthodes (*Software*) et surtout d'organisation spécifique (*Orgware).* Les dispositifs informatiques seraient donc porteurs d'artefacts susceptibles d'agir sur la structure organisationnelle du contexte de travail où ils sont implantés. En ce sens, introduire une technologie, c'est agir sur un système humain, et pas simplement faire tourner un équipement.

En résumé, les approches que nous venons d'exposer nous amènent à considérer les technologies informatiques comme des systèmes qui affectent non seulement les conduites individuelles des informaticiens, mais aussi les aspects socio-organisationnels de leur activité.

b) Le transfert technologique

Quant au transfert technologique, trois définitions le caractérisent :

1) La plus couramment admise est celle d'une coopération technologique et industrielle entre nations (Durand, 1992 ; Wisner, 1996). On parle ainsi de transfert technologique entre pays développés et pays en voie de développement.
2) La notion de transfert de technologie est également utilisée dans l'industrie pour désigner le passage d'une innovation à son application industrielle. (Silem, 1987).

3) Enfin, il peut évoquer la transmission de connaissances technologiques et de savoir-faire (Eyraud, Iribarne, Maurice, 1988), c'est-à-dire « *l'ensemble des compétences, des méthodes, des informations et de l'outillage nécessaires pour fabriquer, utiliser et faire des choses utiles* » (Bizec,1981, p 8).

Si l'on retient cette dernière définition, qui touche plus directement notre thème de recherche, on dira qu'il y aura transfert de technologies informatiques à chaque fois que des dispositifs nouveaux et que leurs savoirs techniques associés sont transmis à des informaticiens. Cela posé, il reste à préciser quelles sont les implications de l'immersion de ces technologies dans le champ de la conception. Sur ce point, deux thèses se dégagent : l'une concerne les implications humaines, l'autre les implications organisationnelles.

4.2 LES IMPLICATIONS HUMAINES DES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE

Si l'on peut transférer facilement des outils et des matériels techniques, il est beaucoup plus difficile d'assimiler les connaissances nécessaires à leur bon fonctionnement. En effet, les définitions que nous avons présentées sur le transfert technologique oublient toutes de préciser que cette transmission se fait rarement *ex nihilo,* c'est-à-dire dans une situation vierge de tout dispositif technique préexistant. Une nouvelle technologie, et c'est encore plus vrai dans le domaine de la conception informatique, vient toujours remplacer un autre système déjà présent et fortement intégré au contexte de travail. Il existe donc toujours un dispositif préalable pour lequel l'informaticien a développé un registre de connaissances spécifiques. Dans ces conditions, le transfert technologique implique fondamentalement des processus d'apprentissage et de réapprentissage qui nécessitent non seulement l'acquisition de nouveaux savoir-faire, mais aussi la maturation et/ou l'ajustement de compétences plus anciennes.

L'étude réalisée par Frese et Hesse (1995) sur le changement d'outils de programmation illustre d'ailleurs cette problématique. Sur la base d'un questionnaire réalisé auprès d'une trentaine de firmes allemandes et suisses, ils ont remarqué que les informaticiens jugeaient la "*facilité d'utilisation*" et "*d'apprentissage*" des nouveaux outils comme des critères plus importants (40% des réponses) que l'obtention de

opératoires entre les deux environnements seront importants, et plus grand aussi sera le risque de rejet du nouveau dispositif.

En somme, ce qu'il faut retenir ici, c'est que le transfert technologique bute fondamentalement sur la question du transfert d'apprentissage et, plus précisément, sur l'adéquation des compétences acquises avec celles requises pour le nouveau système.

4.3 LES IMPLICATIONS SOCIO-ORGANISATIONNELLES DES TRANSFERTS TECHNOLOGIQUES DANS LA CONCEPTION INFORMATIQUE

Un autre problème, rarement abordé dans les approches sur les transferts technologiques en informatique, est l'idée d'un déterminisme technologique où changements techniques et changements socio-organisationnels seraient étroitement liés (Humphrey, 1989 ; Alsène, 1990). Sur ce point, plusieurs thèses s'affrontent :

Pour les uns, une nouvelle technologie n'impose pas en soi un seule type d'organisation du travail, mais en rend possible diverses formes. En ce sens, la rationalisation qui accompagne l'introduction d'une nouvelle technique serait liée aux possibilités ou non d'apprentissage par les individus, et à leur participation ou non au processus de changement technique (Jabes, 1991; Prigent, 1995 ; Marinko, Jenry et Zmud, 1996). Fischer et Brangier (1990) ont, par exemple, montré que la peur d'être remplacé par une machine (ici, un système expert) ou de voir une partie du travail et des décisions intéressantes revenir à la machine justifiaient les raisons du rejet des systèmes informatiques. Cette inquiétude se traduisait dans l'entreprise par un certain nombre de revendications qui conditionnent les stratégies du personnel.

Pour d'autres, aucune technologie n'est neutre lorsqu'elle est introduite dans une organisation (Dezalay, 1995) : « *un certain type de "design organisationnel" accompagne "naturellement" la technologie* » (Alsène, 1990 p 326). Elle peut alors renforcer l'organisation antérieure. Des auteurs comme Léchevin, Le Joliff, Lanoë (1994) ou encore Cavestro (1987) ont montré ainsi que l'introduction d'une nouvelle technologie reproduisait quelques grands principes du taylorisme. Elle accentuait la spécialisation et la centralisation des tâches et procédait par "*algorithmisation*" du travail, c'est-à-dire par une mise en forme des procédures de travail et des séquences

d'opération, diligentée par le nouveau système technique (Cavestro, 1987). Dans ces conditions, le changement technique peut donc être conçu comme un moyen pour restructurer des postes, un service ou l'ensemble d'une entreprise. Il se présente alors comme le « *Cheval de Troie* » de la rationalisation organisationnelle et de l'idéologie managériale (Pavé, 1993).

Enfin, il y a ceux qui considèrent que le transfert technologique conduit à la redéfinition d'un nouvel ordre social. Autrement dit, il peut "déréguler" des pratiques organisationnelles antérieures (en termes de jeux de pouvoir, d'autonomie et de reconnaissance) pour y substituer d'autres formes de réglementation (Alter, 1996). Il est à noter que ce genre de situation engendre parfois des dysfonctionnements importants car les modèles d'organisation qui sont liés aux nouvelles technologies vont devoir coexister avec la pérennisation d'un modèle organisationnel antérieur (Fischer, 1993).

Pour cette seconde approche, donc, le transfert technologique induit des phénomènes de changement organisationnel et d'innovation sociale dans les entreprises. Mais il faut aussi avoir à l'esprit que tout changement technologique ne génère pas uniformément un seul type de modèle organisationnel : c'est l'usage et non pas les caractéristiques intrinsèques de la technologie qui va en déterminer les effets.

4.4 EN RÉSUMÉ

A la lumière de tout ce qui vient d'être dit, le transfert technologique ne doit plus être considéré comme une simple opération de transition technique : il doit être pensé comme un processus de changement et d'innovation dans les organisations. Plus que la simple transposition d'une technique dans un contexte productif, le transfert technologique conduit à des modifications et/ou à des ajustements des structures socio-organisationnelles et humaines préexistantes. Il provoque, bien sûr, des changements de matériels, mais conduit aussi et surtout à la reconfiguration des savoir-faire et des modèles organisationnels en place.

La partie qui va suivre propose d'ailleurs une illustration concrète de ces phénomènes.

5. ÉVOLUTION DES OUTILS ET DES MÉTHODES DE CONCEPTION INFORMATIQUE

Les outils et les méthodes de conception peuvent être vus comme autant de paradigmes qui vont structurer l'espace d'action, de représentation et d'interaction homme-machine : ils contribuent à formuler des règles qui orientent et encadrent les conduites de l'informaticien. Après avoir décrit les principales évolutions de ces techniques de développement, nous verrons dans quelle mesure celles-ci affectent les façons de penser et d'organiser la conception.

5.1 LES DISPOSITIFS TECHNIQUES DE CONCEPTION INFORMATIQUE

Quatre strates techniques ont été retenues pour décrire l'évolution des dispositifs de conception informatique : chacune exige des manières de penser, de représenter et de solutionner les problèmes. Ce sont les caractéristiques : *(i)* architecturales, *(ii)* logiques et fonctionnelles; *(iii)* graphiques; *(iv)*paradigmatiques des langages utilisées

5.1.1 Les caractéristiques architecturales des techniques de conception

Une première analyse nous conduit à opposer les architectures centralisées (site-central) aux architectures distribuées (client serveur).

a) L'architecture site-central: une architecture composée d'un terminal passif relié à un site-central

La plupart des applications informatiques de gestion développées depuis le début des années 70 ont été conçues dans une architecture centralisée, appelée "mainframe" (Desaintquentin et Sauteur, 1991). Cette architecture comprend, d'un côté des terminaux "passifs" (employés par les informaticiens et les utilisateurs) et, de l'autre, un "système central" (le Mainframe) reliés entre eux par un réseau. Dans ce mainframe sont centralisés les programmes et les bases de données qui font tourner les applications.

Si certains mérites sont reconnus à cette architecture :

- *intégrité et fiabilité des données* (contrôle systématique de l'information saisie par rapport à des valeurs autorisées) ;
- *sécurité de fonctionnement* (sauvegarde systématique, robustesse et fiabilité du matériel) ;

- et *confidentialité des accès à l'information* (contrôle strict des accès) ;

elle présente aussi d'incontestables inconvénients :

- *coût élevé de CPU* (unité de mémoire consommée pour chaque transaction sur le site-central) ;
- *complexité de la maintenance des applications ;*
- *absence de convivialité* (interface textuelle médiocre et dialogue en mode transactionnel) avec rejet des utilisateurs.

b) Une architecture distribuée : une architecture composée d'un module client communicant avec un module serveur délocalisé (architecture client serveur)

Le début des années 90 marque l'apparition des architectures client serveur dans les entreprises. Un système client serveur est un système informatique réparti. Le client est un ordinateur requérant des informations ou un service d'un autre ordinateur : le serveur. En général, les clients sont des ordinateurs personnels utilisés pour des tâches de gestion données (Harmon, 1994). Il existe plusieurs aménagements possibles de cette architecture selon l'affectation des données, des traitements et de l'interface aux postes clients ou serveur. Mais la forme la plus généralement observée reste celle-ci : le module client prend en charge la gestion des dialogues entre l'utilisateur et l'application (gestion de l'interface) ; tandis que le module serveur rend le service en question (traitement, calcul, procédures) et gère l'accès aux bases de données.

Dans une architecture client serveur, les perspectives sont donc inversées par rapport au site-central : le terminal passif devient un ordinateur "actif" ou une station de travail "intelligente" car il peut recevoir et réaliser lui-même des traitements.
L'architecture client serveur permet d'échapper au schéma imposé par les applications tournant sur une architecture site-central , avec trois avantages concrets :

- *optimisation des ressources* : le module "client" optimise l'interface avec l'utilisateur en offrant convivialité (graphique) et interactivité, tandis que le module "serveur" se concentre sur les services à rendre (en permettant, par exemple, un accès plus rapide aux donnée et aux traitements) ;
- *réduction des volumes de données à transmettre* (réduction des coûts et des temps de réponse) ;
- *souplesse, adaptabilité et évolutivité* : l'adjonction d'un nouveau service ou d'un nouveau programme ne met pas en péril l'édifice déjà établi.

c) Dans quelle mesure ces évolutions techniques affectent-elles l'activité de l'informaticien ?

Au cours de nos recherches bibliographiques, nous n'avons trouvé trace d'aucune étude qui ait analysé scientifiquement les incidences du changement d'architecture technique sur la conception informatique. Cela dit, certains auteurs se sont interrogés sur les enjeux réels ou cachés du développement des systèmes client serveur dans les entreprises.

C'est le cas de Lasfargues (1995) qui affirme que «*sous couvert de sécurité et de client serveur, on assiste à un formidable retour à l'informatique des années 70* ». L'architecture client serveur ne serait ainsi rien d'autre qu'un système centralisé déguisé. Principal responsable, la peur des virus qui conduit à la centralisation des informations et des applications au niveau des serveurs. Leurs accès sont verrouillés et cloisonnés par la multiplication des clefs d'accès. De même, plus question pour l'informaticien de choisir librement son tableur, son traitement de texte ou un outil de développement particulier. Il doit utiliser les outils que des cellules spécialisées ont testés et validés pour son activité.

Du coup, ces micros bridés redeviennent alors de simples terminaux passifs. Dans le couple "client serveur", le serveur prend en quelque sorte le dessus : il dicte sa loi ; une loi qui ressemble étrangement à celle du "site central" tant décrié autrefois. En somme, pour Lasfargues «*on reconstitue, avec tout un discours client serveur, exactement les grands systèmes d'autrefois* ». Et même si les écrans sont plus séduisants et les applications plus faciles à utiliser, la logique organisationnelle serait la même qu'il y a 20 ans, c'est-à-dire une tendance vers la centralisation.

Dans ces conditions, les mêmes causes produisant les mêmes effets ; en bridant les micros, ce sont les informaticiens que l'on briderait : « *En leur ôtant la liberté d'utiliser leur micro, on risque de tuer l'autonomie et donc la créativité des individus* » (Lasfargues, 1995).

En somme, une solution technique s'accompagne toujours de mesures organisationnelles qui vont façonner et baliser le cadre de la conception.

5.1.2 Les caractéristiques graphiques des techniques de conception

Un autre aspect de ces techniques de conception concerne les interfaces de communication homme-machine. On distingue l'interface textuelle (caractéristiques des architectures site-central) de l'interface graphique (caractéristique des architectures client serveur).

a) Une interface en mode textuel

Le terminal comporte une interface en mode textuel (caractères alphanumériques uniquement) qui est graphiquement et "ergonomiquement" pauvre (Menadier, 1991). Elle ressemble aux écrans du Minitel combinant des zones opaques et des zones ouvertes à la saisie. Le dialogue entre le système et l'informaticien est réalisé à travers les images successives qui apparaissent à l'écran. L'affichage de plusieurs fenêtres en mode multifenêtrage est donc impossible, ainsi que l'utilisation de la souris, la présentation d'icônes, de graphiques et de menus.

```
AGATE                                                    DATE :  XXXXXXXXXX
 - P007 -
                      SAISIE PIECE REGULARISATION
                           DE CATEGORIE C

         COMPTE DE CENTRALISATION  : _________
         DATE DE CENTRALISATION    : __ __ __
         COMPTABLE CENTRALISATEUR  : ____
         CLE                       : __
         CODE VIGNETTE OU AVIS DEP.: ___
         CHAINAGE SERVICE          : ______
         MONTANT                   : ________________

         ETIQUETTE                 : XXXXXXXXXX

                      POUR VALIDER : ENTREE
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   RETOUR MENU    :  PF2
   FIN DE SAISIE  :  PF4
   ABANDON        :  PF3
```

Figure 4
Exemple d'interface en mode textuel (ou alpha-numérique)

b) Une interface graphique

A bien des égards, on peut considérer que l'interface graphique représente l'un des apports les plus fondamentaux de la révolution de la micro-informatique. Elle simplifie l'utilisation et la compréhension de l'environnement en proposant de nombreux dispositifs d'accès intuitifs : multifenêtrage, représentations iconiques, manipulation directe, menus déroulants...

Ainsi, convivialité et simplicité d'utilisation permettent de créer les conditions d'une réelle interactivité entre les outils de conception et les informaticiens (Grudin, 1990). Ces derniers ont l'impression d'interagir "naturellement" avec des objets de leur univers mental, et non avec une machine qui leur impose des contraintes injustifiées (Owen, 1987 ; Palanque et Bastide, 1995).

c) Dans quelle mesure ces évolutions techniques affectent-elles l'activité de l'informaticien ?

Nanard (1991) ; Burnett, Baker, Bohus, Carlson, Yang et Zee (1995) estiment que les informaticiens sont plus enclins à manifester des comportements spontanés de conception (appelés "*directness*") avec une interface graphique, car ils opèrent dans ce qui leur semble être leur propre monde. En outre, comme l'a mis en évidence Schnaider Hufschmidt (1991), l'interface graphique réduit considérablement la charge mentale du développeur en minimisant le recours à la mémorisation des commandes de programmation et des séquences d'action ; ce qui lui laisserait d'autant plus de ressources cognitives pour exercer son art.

Dans un autre genre d'étude, Rauterberg (1996) a tenté de mesurer les qualités ergonomiques de différentes interfaces de conception dotées respectivement d'un dispositif de commandes : *1)* par menu (MI) ; *2)* par icônes et manipulation directe (interface graphique) (DI) ; *3)* par codification (caractéristiques d'une interface textuelle) (Batch) ; *4)* par touches fonctions et raccourcis clavier (proche d'une interface textuelle) (CI).

Pour ce faire, l'auteur comptabilise des "*points d'interactivité*" calculés d'une part, à partir du niveau d'accessibilité de l'interface et, d'autre part, par le type d'information qu'elle délivre suite aux actions de l'individu (*Feedback*). Des évaluations empiriques sont également effectuées. L'auteur compare ainsi la performance d'utilisateurs experts et novices sur chaque interface, en prenant en compte le temps de réalisation d'une tâche, le taux d'erreurs, le nombre d'essais, le temps mis pour corriger les erreurs.

Concernant *les points d'interactivité* des interfaces, Rauterberg présente le tableau de résultats suivant :

		VISIBILITÉ DE L'INTERFACE (FEEDBACK)	
		Bas	**Haut**
ACCESSIBILITÉ DE L'INTERFACE	**Bas**	Interface à codification (Batch)	Interface à Menus (MI)
	Haut	Interface à commandes clavier (CI)	Interface à manipulation directe et icônes (DI)

Figure 5
Niveau d'interactivité des différents dispositifs de communication (selon Rauterberg, 1996)

En clair, les interfaces "Batch" et "CI" offrent une visibilité moins grande que les dispositifs (MI et DI) sur les actions engagées par l'utilisateur. Inversement, les interfaces CI et DI sont les plus accessibles (en rapidité) par leurs dispositifs d'interaction spécifiques. Concernant *l'évaluation empirique* des interfaces, les résultats indiquent que les experts se montrent plus performants que les novices dans l'utilisation des interfaces CI. Toutefois, ces mêmes novices, totalement impuissants sur CI, obtiennent d'excellents résultats avec MI et DI.

Deux critiques doivent cependant être adressées à cette étude :

- d'une part, elle est uniquement descriptive et non explicative. Elle se borne à énoncer les performances des utilisateurs sur les différentes interfaces, mais ne donne aucune explication sur les raisons de ces écarts : habiletés motrices ? automatismes cognitifs ? transferts d'apprentissage plus efficients ? organisation structurée des connaissances ?...
- d'autre part, Rauterberg ne s'est pas du tout intéressé aux performances des différents utilisateurs sur l'interface de type Batch. Seule la comparaison croisée des aspects de surface a été réalisée.

Malgré tout, on retiendra que l'interface graphique "*à manipulation directe*" s'avère la plus ergonomique (meilleure visibilité et accessibilité) et la mieux employée par les utilisateurs novices. A l'inverse, les interfaces "*à codification*" et "*à commande*" se révèlent nettement moins conviviales pour ces mêmes débutants. Dans ces conditions, on peut supposer que l'apprentissage d'un nouveau dispositif de développement serait plus efficient si ce système dispose d'une interface graphique à manipulation directe.

Face à ce très large consensus favorable à la conception graphique, Berlioux (1995) adopte une position discordante. Il affirme, en effet, que ces interfaces graphiques, loin de favoriser la conception informatique, peuvent au contraire nuire à l'efficacité du processus de développement. Il l'explique en avançant l'argumentaire suivant :

- la relative facilité de manipulation des outils de conception peut faire oublier les bonnes habitudes et inciter à développer d'emblée sans concevoir (c'est-à-dire sans réfléchir au modèle à implémenter) ;
- la construction de l'interface relève plus souvent d'une "œuvre personnelle" que d'un produit visant à l'efficacité, à l'homogénéité et à la réutilisabilité. L'informaticien fait ainsi un usage "*spéculaire*" de l'interface ;
- enfin, la conception de la maquette est le plus souvent implicite et non formalisée.

C'est pourquoi, poursuit l'auteur, la présence de normes et de méthodes de conception, édictées par les services compétents, permet d'assurer la cohérence et la fiabilité du développement.

Les critiques soulevées par Berlioux nous amènent à faire la réflexion suivante : bien que l'interface graphique favorise l'adéquation homme-machine, l'informaticien n'est jamais entièrement libre dans le choix de ses actions. Car, lorsque les systèmes informatiques se révèlent impuissants pour encadrer l'activité de l'informaticien, ce sont les structures organisationnelles qui vont prendre le relais en produisant des normes. Nous en verrons d'ailleurs des exemples plus concrets dans les passages ultérieurs de notre étude qui traitent des méthodes de conception. En d'autres termes, l'environnement organisationnel peut, comme pour l'architecture technique, renforcer et/ou canaliser les modalités d'utilisation de ces outils graphiques.

5.1.3 Les caractéristiques logiques des techniques de conception

Une troisième approche concerne la logique de conception propre à chaque dispositif de développement. On oppose généralement la logique transactionnelle *–imposée par les outils site-central–* à la logique événementielle *–proposée par les outils client serveur–*.

a) Une logique de programmation basée sur une logique transactionnelle

Le paradigme transactionnel fait référence à la synchronisation des tâches et des interactions homme/machine. Ainsi, les postes terminaux sont "enchaînés" à l'ordinateur central par le lien du réseau. Ils ne peuvent servir à rien d'autre qu'à exécuter les programmes transactionnels dont sont constituées les diverses applications de production : saisie de données, mise à jour des fichiers de données, traitements... Mais le plus important est que "*tout est prévu à l'avance*" : l'allure des écrans, la manière dont les utilisateurs vont être interrogés et devront formuler leur réponse, les enchaînements d'un écran à un autre... Rien n'est laissé au hasard. L'utilisateur (qu'il soit opérateur final ou informaticien) n'a plus qu'à se laisser guider par des indications qui apparaissent à l'écran. En somme, c'est plus l'ordinateur qui conduit et dirige l'interaction homme/machine que l'inverse. Ces applications obéissent aux règles d'un schéma d'organisation que Desaintquentin et Sauteur (1991) ont baptisé le modèle « **maître-esclave** » : derrière leur poste de travail, les utilisateurs sont totalement dépendants de la logique des transactions imposée par le système. Aucun autre scénario ne peut être envisagé. Le poste de travail est donc un terminal passif, placé à l'extrémité ultime d'un processus totalement prédéterminé et figé.

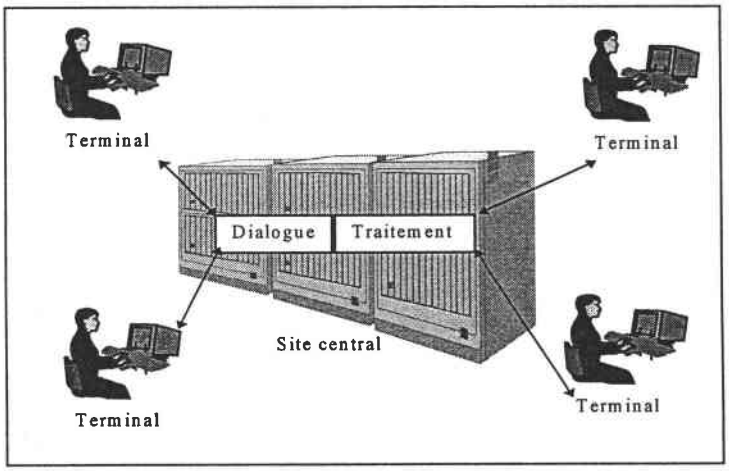


Figure 6
Les interactions homme-machine sont totalement synchronisées par le système central (Modèle maître-esclave).

b) Une logique de conception basée sur une logique événementielle

La logique événementielle peut se caractériser par la liberté d'action laissée à l'utilisateur dans son dialogue avec l'application. Ainsi, le dialogue homme-machine n'est plus figé, ni déterminé *a priori* lors de la conception, comme c'était le cas pour le site-central. Son articulation dépend essentiellement de l'arrivée des données et des objectifs de l'utilisateur. Ce dernier peut effectuer des opérations dans n'importe quel ordre, de manière très dynamique et intuitive (logique asynchrone).

Face au caractère imprévisible de l'interaction, l'informaticien doit alors établir un dispositif capable d'accueillir les requêtes lorsqu'elles arrivent, de les relier au(x) traitement(s) concerné(s) (*par exemple, réaliser simultanément l'affichage d'une fenêtre et la fermeture d'une autre*), et de rétablir toutes les informations utiles dans l'état où elles étaient lors de l'émission du message précédent. Autrement dit, alors que c'était à l'utilisateur de s'adapter à la logique transactionnelle de l'application site-central ; le rapport s'inverse ici : c'est le système qui doit dorénavant s'adapter à la conduite de l'utilisateur.

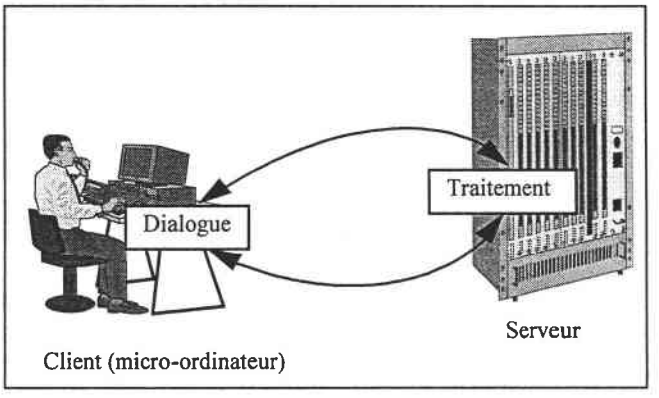


Figure 7 : Modèle client serveur
L'asynchronisme est de règle entre le poste client et le serveur.

c) Dans quelle mesure ces évolutions techniques affectent-elles l'activité de conception ?

Ce changement de logique est sans conteste l'un des plus importants auxquels les informaticiens sont confrontés. Dans une étude effectuée sur l'évolution d'un environnement de conception informatique (de site-central vers client serveur), Bobillier Chaumon et Brangier (1996) ont ainsi montré que cette migration représentait une rupture importante dans les pratiques de travail des développeurs : les stratégies cognitives mises en œuvre pour répondre aux exigences de la conception transactionnelle étaient fondamentalement différentes de celles requises pour

développer en logique événementielle. Concrètement, si la première imposait à l'informaticien de préparer une démarche linéaire et peu innovante, à cause de la logique très prescriptive du système, la seconde recommandait une démarche plus exploratoire afin de mettre au point des solutions de conception originales qui tiennent compte des nombreuses alternatives d'interaction entre l'utilisateur et l'application. A titre d'exemple, l'informaticien devait anticiper toutes les situations d'interaction homme-machine possibles afin de concevoir l'interface.

Dans une autre étude, Brangier et Bobillier Chaumon (1995) ont analysé l'activité de conception d'informaticiens sous deux environnements de développement différents : l'un procédural (outil Pacbase proposant une logique de conception transactionnelle), l'autre orienté objet (environnement smalltalk proposant une logique de conception événementielle). Parmi les résultats de cette recherche, les auteurs indiquent que les scénarios de conception élaborés à partir des indications du paradigme transactionnel se répètent régulièrement –à quelques variantes près– d'un projet de développement à un autre. Le cadre sclérosé de la logique transactionnelle limite toute velléité de conception originale : à un problème ne peut correspondre qu'une solution et une seule, compatible avec les règles édictées par le dispositif. Tandis que pour la logique événementielle, l'informaticien est constamment incité à innover dans chaque projet de développement à cause de la spécificité de chaque utilisateur et de la variabilité des solutions admissibles pour un problème : l'absence de structures administratrices de la conception laisse l'informaticien entièrement libre dans la production et la configuration de ses solutions.

Au final, on retiendra que cette logique événementielle inaugure un paradigme de conception tout à fait inédit dans l'expérience de l'informaticien : il s'agit moins de développer des programmes que de concevoir des situations d'interaction appropriées (Winograd, 1995). Il s'avère aussi que le passage de la conception transactionnelle vers la conception événementielle marque l'abandon d'un "*environnement de programmation*" purement technique pour un "*environnement de conception*" plus fonctionnel, et que cette évolution pose plus profondément le problème de l'adaptation des stratégies cognitives de conception à un nouveau modèle de développement informatique.

5.1.4 Les caractéristiques paradigmatiques des techniques de conception

Enfin, dernières caractéristiques identifiées, celles qui opposent :

- les paradigmes de programmation procéduraux et les "Ateliers de Génie Logiciel" ("AGL" associé aux architectures centralisées), d'une part ;
- aux langages orientés objets (LOO) et aux "langages de quatrième génération" ("L4G" associé aux architectures distribuées), d'autre part.

a) Des environnements de développement standardisés : AGL et langages procéduraux

Un atelier de génie logiciel (AGL) est un environnement de travail intégré qui offre tout un arsenal d'outils couvrant l'ensemble du cycle d'un projet informatique pour construire une application : depuis la définition des besoins et l'analyse jusqu'à la réalisation et la maintenance de l'application (Hammouche, 1993). Cet AGL se compose généralement de "*squelettes de programmation*" (rassemblant des fonctions de développement standardisées préétablies) et de "*grilles de saisie*" (pour la conception des écrans) qui seront présentés à l'informaticien pour développer l'application. Celui-ci est alors invité à remplir les parties de l'écran demeurées modifiables, l'AGL se chargeant quant à lui de générer automatiquement le code machine (cobol) sous forme de programmes. Le développeur ne peut déroger au canevas de programmation que lui impose la machine.

Ces AGL fonctionnent également avec un langage de programmation procédural[11] (cobol) qui a été très familier entre 1960 et 1980. Cobol reste d'ailleurs, aujourd'hui encore, le langage le plus utilisé pour l'informatique de gestion et de production. Son paradigme décrit pas à pas les procédures d'un raisonnement : un petit nombre de verbes (*écrire, lire, calculer, aller, achever, etc.*) suffit à constituer le programme d'une application. Il se prête bien à la programmation des algorithmes[12], en particulier numériques. Un langage procédural se fonde enfin sur la distinction fondamentale des procédures ou traitements et des données qui seront soumises à ces traitements (Pair, 1988).

b) Des outils de conception évolués : les langages de quatrième génération (L4G) et les langages orientés objets

Un L4G représente un ensemble intégré d'outils permettant à la fois de concevoir et de réaliser des interfaces et des maquettes[13]. Plus précisément, les L4G permettent de concevoir, prototyper, exécuter, évaluer et maintenir les applications dans un environnement unique autour d'un formalisme de représentation graphique de l'interface (Wertz, 1993).

Outre les outils classiques de programmation (éditeur de programmes, débogueur...), ces environnements comportent un générateur d'interface graphique qui sert à l'édition des composants graphiques de l'application (menu, fenêtres, boutons...) ainsi qu'à la spécification de leur comportement ("*Fermeture de la fenêtre*", "*Changement d'état d'un menu*"...). La conception se fait par manipulation directe, c'est-à-dire par "*pointage et cliquage*", à l'aide d'une souris, sur des objets graphiques. Ces L4G[14] proposent également des librairies de fonctions informatiques préprogrammées que le développeur peut réutiliser à tout moment dans son projet. On notera enfin, et c'est le plus important, que le L4G se démarque fondamentalement d'un AGL par la relative latitude d'action qu'il laisse à l'informaticien. Ici, plus de squelettes prescriptifs à suivre ou de grilles de saisie à remplir ; l'informaticien demeure entièrement libre du choix de ses actions.

On associe généralement à ce type d'outils un langage de programmation orienté objet (LOO). Le paradigme de programmation OO (C++, Smalltalk) est basé sur les notions de classes, d'encapsulation, d'héritage et de transmission de messages. A l'opposé de la programmation procédurale classique qui sépare données et procédures, l'approche OO supprime cette dichotomie en intégrant données et fonctions de traitements (appelées méthodes) dans des entités nommées "classes". Un objet sera une instance de cette classe et les méthodes seront utilisées pour manipuler ces objets. Par propriété

---

[11] Programmer revient ainsi à « *transformer un énoncé (spécification) qui décrit une fonction, en un programme, c'est-à-dire un texte qui peut être interprété par une machine pour calculer cette fonction* ». (Pair, 1988).

[12] Un algorithme peut être conçu comme un ensemble de commandes permettant la manipulation de formules logiques. On peut comprendre son fonctionnement sans savoir comment la description s'écrira en langage machine, comment ce programme sera converti en une séquence d'instructions par la machine abstraite (Winograd et Flores, 1989).

[13] Une maquette, contrairement au véritable prototype, est une ébauche plus ou moins complète du produit final, qui sert à présenter l'interface de dialogue au client.

[14] Parmi les L4G, on trouve des langage orientés objets (type Smalltalk) et des langages de macro-commandes (type NSDK).

d'héritage, toute classe hérite de la structure et des méthodes de la classe mère. Une des caractéristiques essentielles de ces langages est la réutilisabilité des classes d'objets et des méthodes sur différents projets (Perrot, 1994).

c) Dans quelle mesure ces évolutions techniques affectent-elles l'activité de l'informaticien ?

Les différentes recherches qui ont été effectuées sur les paradigmes de développement ont tantôt insisté sur l'absence de compatibilité homme-machine (cas des AGL par exemple) ; tantôt souligné l'influence qu'ils pouvaient exercer sur les conduites de l'informaticien.

En ce qui concerne le manque de compatibilité homme-machine, Nanard (1990) a remarqué que les nombreuses codifications et procédures de programmation à retenir, lors de l'utilisation d'un AGL (plus d'une centaine), induisaient une charge mentale supplémentaire et conduisaient à une augmentation des erreurs de développement et du temps de formation à l'outil. Mais, elle ne précise ni la nature des difficultés rencontrées, ni leurs origines : saturation de la mémoire de travail ? mauvaise représentation des codifications ? formation insuffisante ou inadaptée à l'outil ?....
Détienne (1990a) a quant à elle réalisé une évaluation ergonomique d'un système de programmation orienté objet (POO) où elle a tenté de mesurer l'adéquation de cet environnement à des utilisateurs potentiels, selon des critères qui ont trait au bon déroulement de leur activité dans des tâches de programmation. Elle a confronté quatre programmeurs expérimentés en langage orienté objet à quatre autres programmeurs débutants en POO (mais avec une expérience préalable, un langage procédural "C"). Elle a ainsi relevé que les stratégies de conception en POO se caractérisaient par une gestion plus opportuniste de l'activité de programmation et le recours plus fréquent à des solutions antérieurement construites. Pour favoriser ce type de conduite, Détienne préconise de construire le système suffisamment souple pour permettre aux développeurs de suivre une démarche empirique et d'articuler plus facilement une représentation déclarative et procédurale du programme.

En ce qui concerne l'influence des paradigmes de conception sur les conduites de conception, plusieurs auteurs ont relevé les corrélations entre le langage employé et l'organisation de l'activité de programmation. Pour Burkhardt et Détienne (1995) par exemple, deux types de guidage s'appliquent : soit un guidage procédural de la

résolution associé aux langages procéduraux : c'est un plan de procédure qui guide la résolution ; soit un guidage déclaratif associé aux langages déclaratifs : c'est alors un plan de propriétés statiques qui guide la résolution. Pour Green (1991), la programmation procédurale exige que l'informaticien énonce d'abord l'ensemble de la structure du programme avant de descendre dans les détails et dans l'implémentation.

Le paradigme OO, de son côté, a souvent été présenté comme favorisant et simplifiant la programmation informatique. Gulliksen et Sandbad (1995) indiquent ainsi que la fonction de "réutilisabilité" proposée par ces langages soulage considérablement la charge mentale de l'informaticien car celui-ci ne perd plus son temps et son énergie cognitive à imaginer des entités informatiques proposées par les librairies. D'autres auteurs, comme Rosson et Gold (1989) et Rosson et Alpert (1991) ont montré que ces langages OO développaient la "*spontanéité créative*" des informaticiens en étant plus proches de leur mode de raisonnement que les langages procéduraux classiques. Les objets conceptuels que les informaticiens manipulent auraient la capacité d'être directement implémentables dans le programme sous la forme d'objets techniques.

En définitive, toutes ces recherches indiquent que les langages OO représentent une alternative de programmation plus efficiente et plus économique (en énergie cognitive et en temps de conception) que les traditionnels langages procéduraux puisqu'ils sont plus compatibles avec le fonctionnement mental de l'informaticien et les caractéristiques de la tâche de conception.

Toutefois, Sperandio (1995) met en garde contre les effets pervers de tels langages. Il pointe sur les risques de "réductionnisme fonctionnel" qui consiste à assimiler l'activité de l'utilisateur à une série de fonctions et d'objets pré-construits. En effet, la commodité de la réutilisation pousserait à récupérer d'emblée des entités de programmation, même si celles-ci ne correspondent pas exactement aux problèmes rencontrés. De même, en mettant à disposition ces fonctions réutilisables, on présuppose implicitement qu'il existe une sorte d'homogénéité entre les différents contextes de travail. Toutes les situations de travail pourraient ainsi être modélisées par ces objets préexistants ; ce qui est quelque peu réducteur.

Même remarque à propos des langages procéduraux. Ils appréhendent l'utilisateur final comme un individu qui déploie invariablement son activité selon une démarche algorithmique et séquentielle. Or, l'expérience acquise en ergonomie montre qu'un

sujet essaie toujours de s'extraire de la règle ou de la procédure prescrite pour définir ses propres marges de manoeuvre (activité réelle). En conséquence, le paradigme procédural ne serait pas en mesure de représenter fidèlement l'activité de l'utilisateur.

Ces différentes études montrent finalement que les paradigmes de programmation affectent les raisonnements de conception : programmer avec des langages informatiques correspond autant à une manière de présenter techniquement les problèmes que de se les représenter mentalement. C'est pourquoi, changer de paradigme de conception implique également de transformer en profondeur ses mécanismes de raisonnement, tant au niveau des représentations du problème, que des stratégies de résolution. Nous y reviendrons plus en détails dans l'analyse cognitive de la conception.

5.1.5 En résumé

Si l'on reconsidère les effets de ces différentes caractéristiques techniques sur l'activité de conception, on arrive aux conclusions résumées dans ce tableau :

CHANGEMENTS TECHNIQUES	IMPLICATIONS SUR LA CONCEPTION ET SUR LE CONCEPTEUR
– Changement d'architectures techniques * *site-central → client serveur*	– Communications plus grandes – Plus de flexibilité et de liberté dans la conception – Absence de contraintes techniques palliée par la mise en place de contrôles organisationnels
– Changement de logiques de conception * *Paradigme transactionnel →→ événementiel* * *environnement de programmation →→ l'environnement de conception*	– Conception axée sur l'interaction et non plus sur la programmation – Conception orientée utilisateur et ne reposant plus sur la logique de la machine – Evolution des modèles cognitifs de développement
– Changement d'interfaces * *Interface textuelle → Interface graphique*	– Compatibilité cognitive homme-machine meilleure – Comportements spontanés de conception – Accessibilité et apprentissage plus rapides avec les interfaces graphiques
– Changement de langages de programmation et d''environnements de développement * *AGL → L4G* * *Procédural → orienté Objet*	– Réutilisation de solutions – Simulation plus aisée – Démarche itérative – Adéquation des univers de représentation (objet conceptuel de l'informaticien et objet technique du langage) – Structuration des conduites cognitives par le paradigme du langage

Figure 8

Tableau de synthèse sur les principales évolutions des dispositifs de conception et sur leurs implications sur l'activité de conception

En somme, chaque environnement de conception se caractérise par des propriétés logiques, fonctionnelles, architecturales et graphiques spécifiques qui laissent supposer que le passage d'un système technique vers un autre implique la transformation et/ou la redéfinition de l'activité de conception, des pratiques cognitives du concepteur et des interactions homme-machine.

Mais ces mutations peuvent également être perçues comme autant d'opportunités offertes à l'informaticien pour redéfinir son rôle dans le processus de conception et affirmer sa prééminence sur le système technique.
En effet, on a vu que, depuis les années 70, la plupart des dispositifs de développement tournant sur site-central étaient caractérisés par leur austérité (interface non ergonomique), leur rigidité (architecture centralisée) et leur logique hautement prescriptive (logique transactionnelle ; canevas de programmation des AGL).
La principale raison était que les concepteurs de ces outils privilégiaient *l'interfaçage entre les machines* (pour une transmission optimisée et sécurisée des données) au détriment de *l'interface homme-machine.* A l'opposé, l'homme (l'informaticien) étant doté d'une formidable capacité d'adaptation, on lui a tout simplement demandé de s'accommoder à une machine qui allait guider et contrôler ses conduites de développement. Dans ces conditions, on peut dire que l'informaticien a tenu une "*position résiduelle*" dans la dyade homme/machine, étant donné que son activité réelle n'avait guère de statut propre face aux lourdes contraintes de la machine (Karsenty, Brézillon, 1995).

Le début des années 90 a été marqué par l'apparition de nouvelles techniques qui ont rendu l'informatique moins lourde, plus souple, et surtout plus séduisante. L'arrivée de la micro-informatique a par exemple favorisé l'émergence d'interfaces plus conviviales, plus près des processus de travail et des représentations mentales des informaticiens. Le poste de travail de l'informaticien s'est ainsi ouvert, offrant des facilités d'utilisation et des libertés d'action, tant au niveau des modes d'interaction homme/machine que dans le déroulement de l'activité de conception. Ces améliorations n'ont été rendues possibles que grâce à l'adaptation des outils de conception aux caractéristiques des informaticiens qui allaient les manipuler. Cette fois, c'est donc l'informaticien qui a été placé au centre de l'interaction

homme/machine, place depuis laquelle il a enfin la liberté de penser les rapports aux objets qu'il manipule et à l'activité qu'il y déploie.

D'autres caractéristiques spécifient tout autant l'activité informatique. Ce sont les démarches de conduite de projet qui vont jalonner l'ensemble du processus de conception.

5.2 LES DÉMARCHES DE CONDUITE DE PROJET EN CONCEPTION INFORMATIQUE

Cette partie est consacrée à la présentation des deux principales démarches qui régissent actuellement la conception, à savoir d'une part, le cycle de vie traditionnel en "V" (associé aux développements site-central) et, d'autre part, le cycle de vie par prototypage (associés aux développements client serveur). L'aspect essentiel de notre discussion portera sur la façon dont ces guides structurent l'activité de conception et façonnent le cadre des relations entre l'informaticien et l'utilisateur. Mais, avant toute chose, il convient de définir à quoi correspond précisément le cycle de vie.

5.1.6 Le cycle de vie du logiciel

a) De quoi s'agit-il ?

La principale fonction du cycle de vie du logiciel est « *de donner l'ordre dans lequel les étapes de développement du logiciel se succèdent, puis de définir quels sont les critères permettant de dire qu'une phase est finie et que la suivante peut être commencée* » (Montero, 1991, p 31). On associe souvent à tort ce *"modèle de cycle de vie"* à la *"méthode de développement du logiciel"*. En fait, cette méthode illustre la façon de progresser au sein de chaque étape, donne les outils, diagrammes et règles de production, pour mener à bien leur étude (par exemple, Merise, Rad[15]...).

b) Les phases classiques du cycle de vie d'un système

Au cours de notre recherche bibliographique, nous avons relevé plusieurs appellations pour désigner des phases de conception identiques du cycle de vie (Russel et Galer, 1987 ; Hoc et Visser, 1988 ; Montero, 1991 ; Larsson et Vaino-Larsson, 1991 ; Soberman, 1992). Le tableau présenté ci-après rappelle ces étapes (avec les noms les plus fréquemment cités) et les fonctions qu'elles remplissent dans le cycle de vie.

[15] RAP : Rapid Application Development)

Phase 1 : Etude préalable, l'avant-projet (niveau de spécification fonctionnelle) – Etape "a" : Investigation initiale – Etape "b" : Evaluation de la faisabilité *Constitution du budget et du planning du projet*
Phase 2 : Etude initiale, l'étude fonctionnelle du projet (niveau d'analyse fonctionnelle) – Etape "c" : Etude de l'existant – Etape "d" : Définition des besoins utilisateurs et construction des divers scénarii possibles – Etape "e" : Evaluation des scénarios solutions *Confirmation de la faisabilité - Choix d'un scénario fonctionnel* *Constitution d'un cahier des charges et de financement*
Phase 3 : Etude détaillée et programmation, l'étude détaillée du projet (niveau conception) – Etape "f" : Conception du nouveau système – Etape "g" : Validation finale des spécifications organisationnelles par les utilisateurs – Etape "h" : Choix des matériels et éventuellement des logiciels *Dossier de réalisation*
Phase 4 : Codage et test (niveau des tests unitaires) *Détection et correction des erreurs dans les programmes*
Phase 5 : Implémentation du nouveau système, intégration, recette finale (jeux de tests) et maintenance (niveau utilisation) – Etape "i" : Livraison du logiciel, conversion de l'ancien au nouveau système et recette finale – Etape "j" : Adaptations et affinage de la solution – Etape "k" : Maintenance

Figure 9
Tableau des principales phases d'un cycle de vie de conception du logiciel

Deux démarches de conduite de projet peuvent découler de cette trame générale de conception. Il s'agit du cycle de vie traditionnel en "V" et du prototypage. Leurs spécificités tiennent principalement aux modalités d'arrangement et de déclenchement de leurs différentes étapes constitutives.

5.1.6 Le cycle de vie traditionnel en "V"

Le cycle de vie en "V" du projet est un processus séquentiel dont les étapes et leur articulation sont rigoureusement définies. Chaque phase possède ainsi ses critères d'arrêt, ses documents et ses sorties à produire. Le cycle de vie commence par des spécifications fonctionnelles et se poursuit jusqu'à la mise en exploitation (Cf. Figure 10). Le passage d'une phase à une autre nécessite que la précédente soit terminée et validée. Enfin, les résultats d'une étape sont utilisés en entrée pour l'étape suivante.

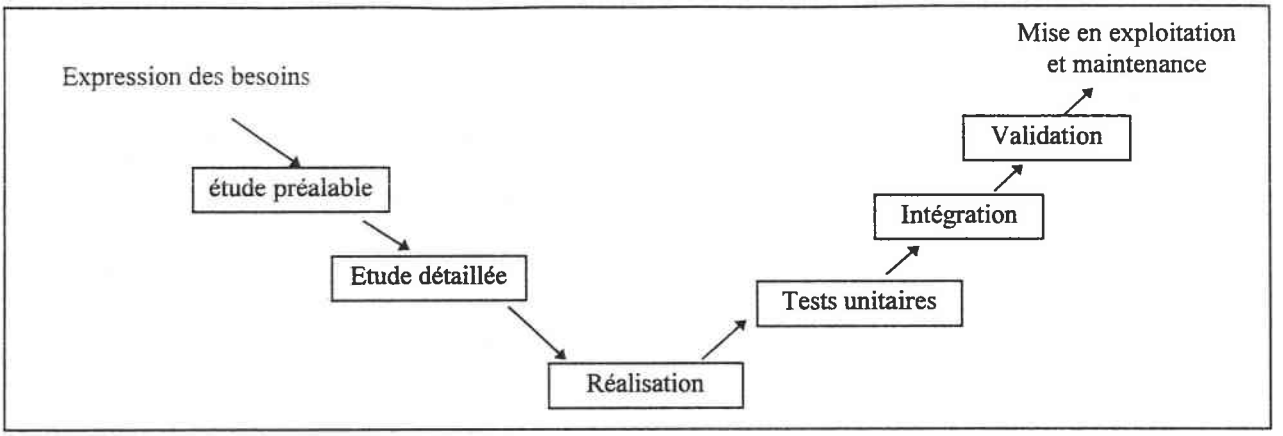


Figure 10
Cycle de vie du logiciel en "V" (d'après Habrias, 1993)

Parmi les études qui sont produites sur ce cycle de vie, la plupart s'interrogent sur les raisons qui peuvent gêner sa mise en œuvre. Les explications les plus fréquemment avancées concernent le rôle de l'utilisateur final dans le cycle de conception et la reconnaissance que lui accorde l'informaticien. Il peut s'agir ainsi :

a) d'une présence insuffisante de l'utilisateur dans les phases préliminaires du cycle de vie ;
b) du manque d'implication de cet utilisateur dans les phases intermédiaires du cycle de vie ;
c) des conséquences d'un "*cercle vicieux méthodologique*".

a) La présence insuffisante de l'utilisateur dès les premières étapes du cycle de vie

Les études réalisées par Rauterberg et Strohm (1992) et Rauterberg, Strohm et Kirsch (1995) ont analysé les démarches de conception employées par un échantillon de 105 services informatiques. Les recueils se sont faits en partie par analyse de document et entretiens (pour 22 d'entre eux) et en partie par questionnaires (pour les 83 restants). Ils constatent que les utilisateurs sont non seulement très peu impliqués dans les phases préliminaires de la conception, mais qu'en plus, ces phases sont souvent déconsidérées par les informaticiens, par rapport au reste du cycle de vie (Rauterberg et al., 1995). Ce qui contraste avec d'autres résultats démontrant justement que plus les efforts sont concentrés dans ces étapes de préparation de l'activité, moins les coûts et les efforts de récupération des erreurs sont élevés (Rauterberg et Strohm, 1992). Les auteurs remarquent enfin que trois types de problèmes accompagnent systématiquement la mise en œuvre du cycle de vie en "V" :

1. *Le problème de la spécification* : Il existe toujours une part d'incertitude, plus ou moins grande, relative à la fiabilité et la stabilité des besoins exprimés par les utilisateurs (pérennité des besoins). Sont-ils suffisamment préparés ? Maîtrisent-ils un minimum le vocabulaire fonctionnel et/ou technique leur permettant d'expliquer correctement ce qu'ils attendent ? « C*ela serait une grave erreur de présupposer que les utilisateurs –souvent des personnes d'un niveau intermédiaire de l'encadrement– sont capables de fournir des informations appropriées et adéquates concernant leurs besoins en matière de développement de logiciel interactif »,* affirment Rauterberg et Strohm (1992).

2. *Le problème de la communication* : Autre problème, la barrière de communication entre les développeurs et les utilisateurs qui ont des savoirs, des expériences et des référents techniques, culturels, historiques et sociaux souvent différents. Pour cette raison, l'utilisateur ne sait jamais si les besoins qu'il émet trouveront un écho favorable chez l'informaticien, c'est-à-dire si ce dernier réinterprétera convenablement ses demandes. Pour Rauterberg et Strohm (1992), seule la réalisation effective des besoins en temps réel (par le maquettage) permet de confronter les représentations de chacune des deux parties, de les ajuster et finalement, de briser ce mur de la communication. Cette idée rejoint celle de Corniou et Hattab (1996) pour qui l'utilisateur tente désespérément de formuler ses besoins dans des concepts abstraits qui ne sont pas les siens, mais ceux de l'informaticien. Ces tentatives d'explicitation se soldent souvent par un échec et un abandon au profit du "professionnalisme" de l'informaticien. Finalement, passablement frustré et fatigué par tant d'efforts non récompensés, l'utilisateur en viendra à considérer toute automatisation comme une technique agressive ne respectant pas son point de vue personnel

3. *Le problème de l'optimisation* : Dernière difficulté, celle qui consiste à croire que l'optimisation du cycle de conception du logiciel n'est possible que grâce aux nouvelles méthodes et techniques de développement. Les auteurs affirment au contraire que l'attention apportée aux étapes initiales du cycle de vie, et en particulier aux phases d'analyse, d'évaluation et de planification du processus du développement, suffit à garantir le succès du projet. Seulement, les

entreprises préfèrent investir dans des technologies nouvelles et se concentrer principalement sur la phase de programmation au détriment des autres étapes, comme le montre ce graphe réalisé à partir de questionnaires soumis aux entreprises (Cf. Figure 11).

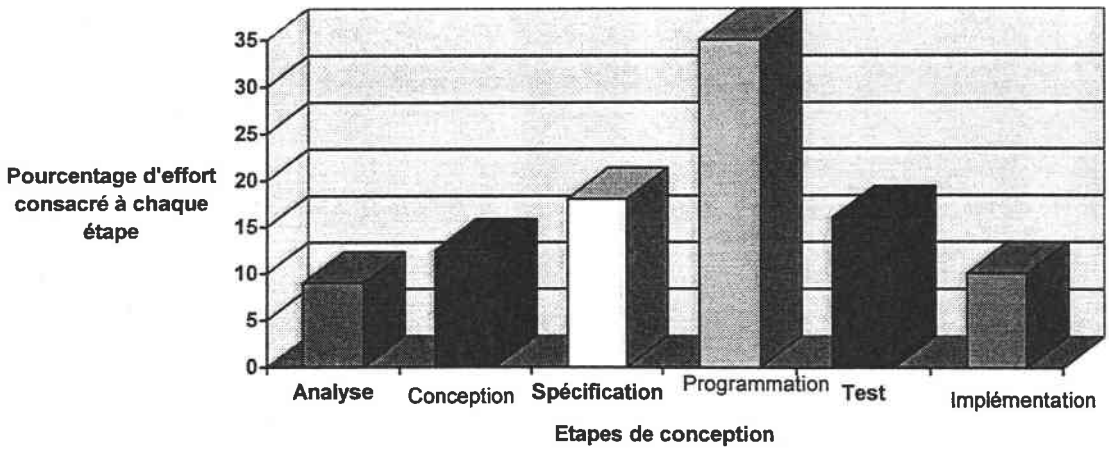


Figure 11
"Echelle" des efforts (en terme de charge de travail) consentis par les entreprises dans les différentes étapes de la conception (sur 79 entreprises)

En somme, pour cette première série d'études, les principales difficultés du cycle en "V" proviendraient de l'insuffisante prise en compte de l'utilisateur dès les premières étapes du cycle de vie. A ceci, s'ajouterait la croyance selon laquelle les nouvelles technologies permettraient de gommer les carences et les imperfections méthodologiques. Pour d'autres chercheurs, au contraire, les difficultés de conception trouvent leurs origines dans le manque de coopération informaticien/utilisateur au cours des phases intermédiaires du cycle de vie, c'est-à-dire lors du développement et du maquettage.

b) La présence insuffisante de l'utilisateur dès les étapes intermédiaires du cycle de vie

Dans une série d'études effectuées par entretiens semi-directifs auprès d'une dizaine de personnes et par questionnaires sur 103 services informatiques, Agro, Cornet et Pichault (1995b) ont répertorié les différents niveaux d'implication de l'utilisateur dans la conduite d'un projet (Cf. Figure 12) :

Analyse des besoins	83%
Analyse fonctionnelle (analyse des tâches à informatiser)	65%
Organisation du travail	53%
Définition du contenu de l'application	48%
Choix des tâches à informatiser	47%
Ergonomie	31%
Evaluation de l'application	29%
Evaluation de la maquette	23%
Elaboration de la maquette	22%
Choix du logiciel	11%
Choix du matériel	4%

Figure 12
Niveaux d'implication des utilisateurs dans la conduite de projets

Les résultats de cette étude fournissent des indications très intéressantes sur la manière dont le client est intégré au processus de conception. Ainsi, s'il semble que ces utilisateurs soient massivement impliqués en amont du projet, lors de la définition des besoins (83%) et de l'analyse fonctionnelle (65%), cette participation ne cesse de régresser pour tomber à des scores plus faibles de 11% ou 4%, lors des choix de matériels ou de logiciels. Tout se passe donc comme si cette participation était ramenée à une simple formalité ou obligation contractuelle, point de départ du processus de développement informatique qui, une fois terminée, conduit les développeurs à réaliser d'autres étapes où les utilisateurs n'ont plus à se prononcer (par manque de compétences ou par crainte d'une prise de pouvoir sur le développement).

L'implication de l'utilisateur se résumerait ainsi souvent à une simple consultation, opérée une fois pour toutes en amont du processus, c'est-à-dire à un moment où les utilisateurs concernés n'ont pas encore pu apprécier les enjeux et les retombées de la nouvelle application sur leur travail quotidien. Le risque est qu'une telle prise de conscience intervienne beaucoup plus tard et qu'elle soit à l'origine de corrections coûteuses ou du rejet pur et simple du projet.

Dans cette seconde approche, l'utilisateur est donc écarté pendant la seconde moitié du cycle de développement. L'informaticien se présente comme l'unique architecte de la construction du logiciel. Il assume seul les choix de conception qui affecteront pourtant les conditions d'utilisation de l'application par l'opérateur final. Ce dernier n'est alors qu'un simple "spectateur" de la conception, chargé de valider le travail

présenté par l'informaticien. En fin de compte, ce manque d'implication débouche assez souvent sur le rejet du nouveau système qui est perçu comme un corps étranger venant se greffer sur l'activité quotidienne de l'utilisateur. D'où une troisième cause des difficultés inhérentes au cycle de vie ; le "cercle vicieux méthodologique".

c) Le cercle vicieux méthodologique

Dans le même type d'étude que précédemment, Agro, Cornet et Pichault (1995b) ont tenté d'identifier les phénomènes à l'origine des tensions entre les informaticiens et les utilisateurs (Cf. Figure 13).

Changement dans l'organisation du travail	58%
Difficultés au moment du démarrage de l'application	45%
Surcharge de travail pour les utilisateurs	37%
Manque de formation et/ou d'information	33%
Retards par rapport aux délais annoncés	28%
Lenteur de l'application	18%
Contrôle accru sur le travail des opérateurs	18%
Menace pour l'emploi	16%
Manque de convivialité de l'application	8%

Figure 13
Principales causes des conflits opposant informaticiens et utilisateurs

Ce sont principalement les changements organisationnels provoqués par l'implantation d'une nouvelle technique (58%), viennent ensuite le démarrage et l'apprentissage de l'application (45%) et la surcharge de travail engendrée par celle-ci (37%). En somme, ces conflits surviennent à chaque fois que l'utilisateur estime que le nouveau système est à l'origine d'une rupture dans ses façons de raisonner, d'organiser et de gérer son travail. Mais ce qu'il est intéressant de relever ici, ce sont les conséquences insidieuses que ces difficultés génèrent sur la collaboration informaticien-utilisateur : tout se passe en fait comme si les critiques et les résistances qui apparaissent chez des utilisateurs devant les systèmes défectueux suscitaient l'ire des responsables informatiques de projet. Lesquels, estimant avoir "tout" mis en oeuvre pour répondre le plus précisément possible aux besoins des utilisateurs, allaient tâcher de s'en débarrasser pour les projets ultérieurs, et ce, dans le but d'éviter de nouveaux conflits. Ce qui aurait pour effet d'exclure encore davantage l'utilisateur de la conduite de projet et de créer les conditions d'un "cercle vicieux méthodologique".
En définitive, toutes ces études ont le mérite d'avoir considéré l'activité du concepteur à travers le cadre de résolution formé par le cycle de vie traditionnel. Elles ont permis

de relever un certain nombre de problèmes qui pouvaient se poser à l'occasion de la conception, et notamment ceux engendrés par l'éviction de l'utilisateur dans le processus de développement. On a vu que cet éloignement n'avait pas seulement une origine structurelle (inhérente à l'organisation même de la démarche) mais qu'elle découlait aussi d'un phénomène conjoncturel ; l'informaticien collaborant opportunément avec l'utilisateur pour maintenir son pouvoir.

Si cette démarche de conduite de projet s'est révélée particulièrement bien adaptée pour le développement des applications site-central. l'irruption des systèmes client serveur a nécessité, en revanche, une plus grande souplesse d'action dans la programmation. En particulier, de nombreux allers-retours entre les différentes étapes de développement pour réaliser les réglages et des tests. C'est pour cette raison que, dès le début des années 90, une nouvelle démarche s'est déployée : le prototypage.

5.1.6 Le cycle de vie itératif par Prototypage

Ce cycle de vie est un processus itératif au cours duquel se succèdent des phases d'affinements progressifs des spécifications du produit, d'évaluation des solutions retenues, de réalisation et d'intégration des parties (Cf. Figure 14). Mais sa grande particularité par rapport au cycle traditionnel est que la spécification d'un ensemble de composants à un niveau donné peut entraîner la remise en cause de certains choix faits au niveau précédent (Michard, 1993).

Cette propriété répond à deux exigences : d'une part, réduire les délais de production du logiciel (Hoyos, Gstalter, Strube, Zang, 1987) et, d'autre part, repenser le cycle de vie d'un projet en intégrant, dans les différentes phases de conception et de fabrication du logiciel, le point de vue de l'utilisateur final (Harker, 1987). Les nouveaux outils graphiques de conception (générateurs d'interfaces) ainsi que les langages de programmation orientés objets contribuent d'ailleurs largement à cette approche par prototypage. Ils permettent d'effectuer les simulations des solutions retenues en réalisant des maquettes[16] et de les tester auprès des utilisateurs (Chao, 1987).

[16] Une maquette correspond à un enchaînement d'écrans avec des données truquées. Elle comporte un minimum de codes (assez pour déclencher les événements) et une présentation graphique relativement complète de l'interface.

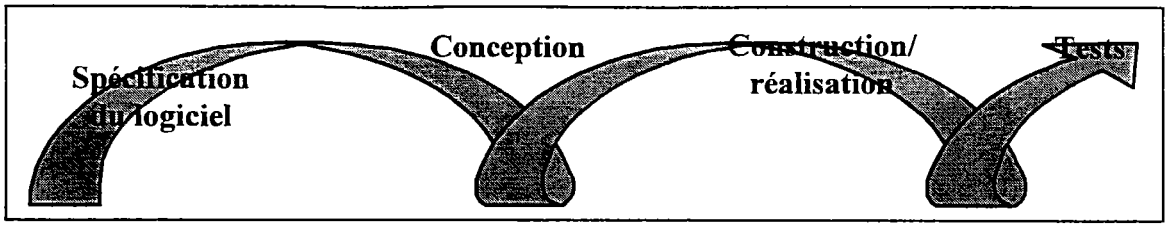


Figure 14
Conception en spirale ou par prototypage

Plusieurs mérites sont reconnus à cette méthode, notamment (Montero, 1991) :

- le fait d'empêcher un certain nombre de problèmes inhérents aux premières versions d'un logiciel. Par son exécutabilité immédiate, le prototype peut offrir une première validation du futur système logiciel, dès la phase d'analyse ;
- cette approche s'adapte bien aussi à des spécifications floues ou changeantes et permet, par sa nature itérative et incrémentale, la capture et l'adaptation graduelle aux besoins exprimés par l'utilisateur ;
- le prototypage est une démarche plus perméable et plus rigoureuse, face aux erreurs : le processus itératif de sa construction permet une rectification postérieure ;
- enfin, c'est une méthode qui implique activement l'utilisateur dans le processus d'analyse-conception, et dont les effets se mesurent par la motivation et l'engagement de l'utilisateur au processus de changement qui le concerne.

En somme, d'aucuns voient dans cette nouvelle démarche la réponse aux nombreuses défaillances du cycle de vie traditionnel. Nous estimons pourtant que ces points de vue sont partiels dans la mesure où ils ne considèrent les améliorations que du côté de l'utilisateur. La transformation de ces méthodes de conception modifie également profondément la place et le statut de l'informaticien.

- En particulier, Fresse et Hesse (1995) ont observé qu'une double pression peut s'exercer lorsque les nombreuses sollicitations des utilisateurs finaux viennent parasiter l'activité de l'informaticien. C'est à cette occasion que se développe ce qu'ils appellent le "*syndrome d'épuisement mental*". Il implique « *qu'une personne se sent fatiguée physiquement, qu'elle n'est plus capable de s'identifier avec son travail et qu'elle a le sentiment que son idéal, ses idées ou*

son potentiel ne peuvent plus se réaliser » (p. 10). Cet état conduit à la baisse de la motivation et de la productivité, ainsi qu'à l'augmentation du taux d'absentéisme et de troubles psychosomatiques (baisse de la vigilance, susceptibilité...).

- Une seconde difficulté provient du fait que les informaticiens ont l'impression que leur projet leur échappe : il y a moins de place pour l'innovation car les utilisateurs contrôlent de très près le développement, au niveau des coûts et des délais (Cohendet, Lherena, et Mutel, 1992). En somme, si auparavant le pilotage du projet était exclusivement l'apanage des informaticiens qui imposaient aux clients leur choix de programmation, aujourd'hui, les services informatiques sont amenés à produire ce qu'ils pensent "vendre" et à ajuster leurs prestations aux exigences des clients (Hammer et Campy, 1993).

- Une dernière raison est liée aux conditions et aux formes mêmes de la participation. Trop souvent en effet, les chefs de projet adoptent un style de management que d'Agro, Cornet et Pichault (1995) qualifient de "*panoptique*", c'est-à-dire de recherche d'une transparence accrue, d'une formalisation croissante de la vie organisationnelle, d'une élimination des pratiques occultes par des techniques, comme l'analyse de la valeur, la certification et autres démarches de formalisation de l'activité. Seulement, une telle approche nie généralement l'existence d'utilisateurs multiples ayant des intérêts divergents et potentiellement contradictoires : les conflits sont alors considérés comme les symptômes de dysfonctionnement qu'il s'agit d'occulter, à défaut d'y remédier (Pavé, 1989). On débouche ainsi sur des formes d'implications procédurières où la participation des utilisateurs est vue essentiellement sur un mode consensuel, lequel est censé favoriser l'acceptation et la participation au projet (Friedberg, 1988). Or, toute formule participative qui chercherait à éliminer les divergences d'intérêts et à favoriser l'acceptation, sur un mode consensuel, d'un projet informatique risque à plus ou moins long terme de se heurter aux résistances issues des rapports de force entre groupes d'acteurs. En d'autres termes, les oppositions se révéleront avec d'autant plus d'intensité en fin de projet qu'elles ont été éludées ou étouffées en début de projet.

Finalement, ces dernières critiques apportent un sérieux bémol au succès du prototypage. On se rend compte que la participation, principal atout du cycle de vie en spirale, rend en fait explicite les divergences de points de vue entre développeurs et utilisateurs, en instituant une arène officielle où elles viennent très logiquement s'exprimer. L'implication et la participation dans les projets ne seraient donc pas les vecteurs du consensus revendiqué, bien au contraire. Elles provoqueraient l'érosion des zones de compétences de l'informaticien et remettraient en cause son pouvoir dans la conception.

5.1.6 En résumé

La conception informatique peut donc être analysée tantôt comme une activité structurée, hiérarchisée et séquentielle lorsqu'elle est régulée par le cycle de vie traditionnel, tantôt comme une activité ouverte, flexible et opportuniste lorsqu'elle s'effectue dans le cadre du prototypage.

De même, chacune de ces démarches définit le cadre des relations entre l'informaticien et l'utilisateur : si, dans un cas, le cycle traditionnel a permis à l'informaticien d'exercer son autorité dans la conception ; dans l'autre cas, le cycle en spirale a inversé la balance des pouvoirs en positionnant l'utilisateur comme l'acteur principal du projet. En définitive, ce changement de méthodes a profondément modifié les perspectives d'approche de la conception : au "*développement centripète*", axé principalement sur les solutions techniques, succède maintenant un "*développement centrifuge*", ouvert au monde extérieur et basé sur la compréhension de l'utilisateur final et des situations de travail.

Au-delà de ces conclusions, on peut aussi dire que ces méthodes offrent bien plus que de simples trames d'organisation pour la conception. Elles mettent à disposition des registres de comportements à partir desquels les acteurs (informaticiens, utilisateurs) peuvent exprimer leur légitimité et leur pouvoir dans la conception. Autrement dit, ces méthodes conditionnent autant les modalités de la coopération qu'elles affectent les formes d'organisation du développement. Elles représentent des matrices qui régulent les conditions d'action et d'interaction entre les pairs.

6. CONCLUSION

Dans ce chapitre, nous avons tenté de fixer le cadre de notre recherche ; à savoir la conception informatique et ses transferts technologiques.

La conception a été présentée comme un processus de développement global qui articulait différentes phases, dans le but de réaliser des dispositifs virtuels. La discussion engagée autour des effets du transfert technologique a permis de dégager deux types d'impacts majeurs : d'une part, les changements techniques induisent des transferts d'apprentissage ; d'autre part, il existe un déterminisme technologique qui définit les manières de penser, d'organiser et de coopérer dans la situation de travail.

Ainsi, les caractéristiques des techniques de conception utilisées (architecturales, graphiques, logiques et paradigmatiques) représentent autant de paradigmes susceptibles d'orienter et d'encadrer les conduites de conception de l'informaticien. De même, les méthodes de conduite de projet (cycle classique en V et prototypage) constituent des matrices qui régulent les modalités d'action et d'interaction de l'informaticien avec son environnement : d'une part, elles prescrivent des configurations d'organisation de la tâche (*organisation hiérarchisée avec le cycle traditionnel en V - organisation itérative avec la méthode en spirale*), d'autre part, elles médiatisent les relations entre l'informaticien et l'utilisateur (en donnant le pouvoir tantôt à l'un, tantôt à l'autre).

En définitive, il apparaît que la transition d'un système technique de conception vers un autre correspond à une situation d'acquisition mentale, résultant à la fois du déterminisme technologique des nouveaux dispositifs, de l'expérience professionnelle acquise, et des stratégies individuelles et collectives fondées à partir de ce que les individus pensent de ce qu'ils font ou ont à faire face à ces changements.

La partie qui va suivre a pour but d'identifier les processus cognitifs qui se manifestent dans les situations de conception informatique et de transfert d'apprentissage.

CHAPITRE II

LES ASPECTS COGNITIFS DE L'ACTIVITÉ DE CONCEPTION INFORMATIQUE

Résumé

Ce chapitre présente les aspects théoriques et conceptuels de l'activité de conception informatique sur lesquels nos travaux sont basés.

La première partie expose les différentes approches de l'ergonomie de la programmation en tant qu'elles fournissent, par leurs études expérimentales et empiriques, des théories sur le fonctionnement humain dans le cadre de la conception informatique

Ensuite, nous présentons les principales caractéristiques de l'activité de conception. Nous verrons en particulier qu'elle se définit comme une activité de résolution qui s'opère sur une catégorie de problèmes bien déterminée ; les problèmes de conception. On distingue aussi la conception routinière de la conception non routinière car chacune réclame un certain type de résolution : créativité Vs routine.

Enfin, les processus cognitifs impliqués dans cette activité de résolution sont examinés : de la compréhension du problème jusqu'à l'implémentation de la solution retenue, en passant par la construction/réutilisation d'une solution, sa planification, et sa correction.

Nous décrivons aussi les mécanismes impliqués dans le transfert d'apprentissage, en particulier la réutilisation de schémas de connaissances et le raisonnement analogique.

1. INTRODUCTION

Dans le chapitre précédent, nous avons présenté les aspects généraux de la conception informatique. Ici, notre intention est d'appréhender l'activité de conception par les processus cognitifs qui l'animent.

Nous avons aussi montré que les transferts technologiques impliquaient fondamentalement des phénomènes de transfert d'apprentissage : ces changements constituent une période de basculement d'un système de travail vers un autre où les acquis professionnels antérieurs doivent être remaniés en fonction de nouvelles exigences cognitives et opératoires. Du coup, se posent des questions plus spécifiques relatives à la nature des compétences et des mécanismes cognitifs en œuvre dans ces transferts.

Nous commencerons par présenter les différentes perspectives de recherche qui ont été adoptées par l'ergonomie de la programmation pour définir et étudier l'activité de conception. Ensuite, nous nous attacherons à décrire les spécificités de cette activité, en montrant qu'elle est un cas particulier de la tâche de résolution de problème. Les étapes cognitives de la résolution informatique seront également exposées de manière exhaustive. Enfin, nous terminerons ce chapitre par la description des transferts de compétences.

2. LE POINT DE VUE DE L'ERGONOMIE DE LA PROGRAMMATION

Concevoir en informatique, c'est programmer à l'avance des séquences de comportements qui affecteront les pratiques des futurs utilisateurs. C'est aussi guider et/ou encadrer les initiatives de raisonnement de ces usagers. C'est encore structurer les modalités d'interaction entre l'homme et la machine, et entre les hommes. C'est enfin conduire les utilisateurs à développer des stratégies opératoires d'appropriation du nouvel outil. En somme, concevoir correspond tout autant à une mise en forme du monde et des conduites humaines, qu'à la prescription des rapports entre les individus et entre l'homme et la machine.

On perçoit donc tout l'enjeu du processus de conception qui, s'il est mal engagé, peut engendrer de profonds bouleversements sur la situation à informatiser. D'où l'importance des conditions de travail dans lesquelles s'accomplit cette tâche, et de la compatibilité des moyens –techniques, fonctionnels et organisationnels– mis à disposition de l'informaticien pour programmer. C'est dans cette perspective d'amélioration des conditions de travail que l'ergonomie de la programmation s'est développée. Elle représente une sous-discipline de l'ergonomie de l'informatique et des systèmes techniques qui va être rapidement présentée.

2.1 ERGONOMIE DES SYSTÈMES TECHNIQUES ET DE L'INFORMATIQUE

Au sens large, cette ergonomie a pour objet de contribuer à l'amélioration des outils informatiques, tant matériels que logiciels, et de leur condition d'utilisation. Dans ce courant disciplinaire, on distingue l'ergonomie des logiciels de l'ergonomie de la programmation. L'ergonomie des logiciels a pour objet l'optimisation de l'interface entre l'ordinateur et l'utilisateur sur les aspects qui relèvent des programmes (Sperandio, 1987). L'optimisation des logiciels eux-mêmes, considérés non du point de vue des utilisateurs finaux, mais de celui des informaticiens (programmeurs) qui les conçoivent, les écrivent, les modifient... fait également partie de l'ergonomie des logiciels, mais on l'appelle plus couramment l'ergonomie de la programmation, dont l'objet d'étude concerne en particulier les aspects cognitifs et coopératifs de la conception de programmes informatiques.

Plus concrètement, cette approche étudie les situations de conception informatique dans le but de permettre la meilleure compatibilité possible entre les informaticiens, leurs tâches (l'activité de programmation) et les dispositifs techniques (langages, logiciels, outils), afin de prévenir les défaillances du système homme-machine et de garantir un dialogue adapté aux situations de travail (Sperandio, 1988). En ce sens, elle peut être décrite comme un cas particulier de l'adaptation du travail à l'homme : l'adaptation du système informatique à l'intelligence humaine (Wisner, 1987). Mais elle apporte aussi, par ses recherches, *« des connaissances sur la manière dont fonctionne un sujet donné sur le plan intellectuel dans une classe de problèmes donnés »* (Bisseret, 1985).

Pour comprendre la spécificité de cette discipline, il est nécessaire de rappeler sa genèse à travers les différents courants théoriques et méthodologiques qui l'ont traversée.

2.2 GENÈSE DE L'ERGONOMIE DE LA PROGRAMMATION

La récente histoire de l'ergonomie de la programmation est un exemple d'une évolution positive des cadres théoriques et des méthodologies d'approche de la situation de travail qui ont suivi de très près la modernisation des techniques de conception et des théories cognitives sur l'homme. Les chercheurs (Hoc et Visser, 1988 ; Rogalski, Samurçay et Hoc, 1988) déclinent l'évolution de cette discipline selon quatre grandes étapes :

a) Elaboration de batteries de tests (années 1960-70)

La première étape pourrait être qualifiée de préhistoire de l'ergonomie de l'informatique. Ce n'est qu'à partir du moment où les coûts des logiciels se sont avérés importants en regard de ceux du matériel que les questions liées à l'ergonomie de la programmation ont été directement abordées. Dans les premiers temps, ce sont surtout des psychologues qui intervenaient en se contentant de construire des batteries de tests de sélection assez peu efficaces par manque d'analyse de l'activité de programmation. Les pratiques de programmation de l'époque étaient en effet très peu standardisées. Les programmes étaient difficiles à comprendre, en ce sens qu'ils étaient davantage marqués par les contraintes liées à une utilisation économique de la machine que par la structure des données traitées.

b) Evaluation empirique des outils de programmation (années 1970)

La seconde étape de l'ergonomie de la programmation s'est traduite, au cours des années 70, par l'adoption d'une attitude très empirique qui a conduit à mettre en oeuvre une méthodologie expérimentale pour évaluer les outils disponibles dans des situations de complexité réduite : aides à la correction des erreurs dans les programmes, caractéristiques des langages existants, des modes de présentation des programmes. Ces réflexions normatives ont conduit à développer les premiers outils (langages, méthodes, aide à la programmation) qui en reflétaient les principes.

c) Etude expérimentale de l'activité de programmation (début des années 1980)

Cette période marque le développement d'une réflexion théorique et méthodologique sur les travaux expérimentaux. On pointe notamment sur les difficultés de généraliser ces résultats du fait :

1. d'abord, d'un manque de prise en compte de la relation entre les situations expérimentales et les situations réelles de travail (représentativité des sujets autant que des tâches choisies),
2. ensuite, des difficultés d'interprétation des résultats en termes de mécanismes cognitifs, du fait du choix de variables de performances assez peu indicatrices de ces mécanismes,
3. enfin, de la nécessité de décrire ces mécanismes pour permettre d'établir des équivalences plus pertinentes entre situations sur des critères de structure cognitive au lieu de simples critères de surface.

d) Insertion des sciences cognitives dans l'activité de programmation (depuis la fin des années 80)

À la suite de ces critiques, une quatrième étape s'est donc engagée. Elle consiste en une approche[17] plus résolue de la programmation par l'intermédiaire de la psychologie cognitive. On s'appuie ainsi sur les stratégies de résolution de problème ou sur le modèle de la "théorie des schémas" pour décrire les stratégies de conception de programme (Détienne, 1988). Dans une perspective pédagogique et d'apprentissage, la

[17] L'INRIA a d'ailleurs développé un secteur de recherche important dans le domaine de la psychologie-ergonomique de la programmation. Les recherches qui y sont conduites portent, entre autre, sur des modélisations de l'activité de programmation (individuelle et collective), sur les conditions de performance de la programmation (avec différents paradigmes et lors de différentes étapes de la programmation), sur l'adaptation des outils de développement aux caractéristiques de leurs utilisateurs et sur l'apprentissage de ces langages.

compréhension des langages et le transfert de connaissances entre les langages intéressent aussi les chercheurs. Enfin, des analyses ergonomiques effectuées en situation effective de programmation vont donner la possibilité aux ergonomes de spécifier des environnements de développement réellement adaptés aux caractéristiques des informaticiens et de leur tâche (Détienne, 1989a, 1989b, 1990c ; Hoc, 1982 ; Hoc et Wisser, 1988).

2.3 UN APERÇU DES DOMAINES D'INTERVENTION EN ERGONOMIE DE LA PROGRAMMATION

Dans le domaine de la conception informatique, l'ergonomie peut avoir quatre types de contribution principale : 1) elle s'attache à rendre compatibles les méthodes et techniques de développement avec l'activité et la logique de raisonnement des concepteurs ; 2) elle contribue à spécifier des assistances aux activités de programmation ; 3) elle favorise aussi, par ses études et ses analyses, l'apprentissage de nouveaux langages informatiques ; 3) elle se propose enfin d'optimiser la collaboration entre les partenaires de la conception.

a) Les méthodes et les techniques de développement

A première vue, la programmation paraît une activité très simple, et pourtant, les programmes sont longs à mettre au point. Il y subsiste des erreurs qui coûtent très cher, à trouver et à corriger. Chaque développeur expérimenté dispose, sinon des solutions, au moins de manières de faire pour limiter les dégâts. L'ergonomie de la programmation étudie ces stratégies (Rogalski, Samurçay, Hoc 1988, Chauvin, 1991).

A titre d'exemple, Sperandio (1988) présente différentes techniques utilisées pour faciliter la lisibilité d'un programme, et donc sa compréhension, lors des tâches de correction et d'évolution de codes :

- la dénomination des variables et des fonctions. La façon de choisir les dénominations et de les abréger a une incidence très importante sur leur identification ultérieure par le programmeur lui-même ou par d'autres programmeurs ;
- les commentaires insérés par les programmeurs dans le programme sont sans incidences sur l'exécution. Ils n'ont qu'une valeur informative afin de faciliter la lisibilité et la compréhension du programme. L'expérience montre pourtant

qu'ils ne sont pas toujours judicieusement employés, même lorsqu'ils sont abondants : par exemple, l'informaticien commente davantage les points qui ont posé problème lors de l'écriture du programme que ceux qui soulèveront des difficultés aux futurs développeurs ;

- le paragraphage, c'est-à-dire l'insertion de titres, de lignes séparatrices, etc. facilite la lisibilité du programme en soulignant la structure des différents segments de programmes ;
- la documentation associée à un programme contribue également fortement à la compréhension des programmes. Les schémas fonctionnels, les textes explicatifs, les commentaires, les annexes... offrent différents angles de lecture du programme, d'autant plus nécessaires que l'équipe projet peut être importante et la durée de vie des programmes longue ;
- etc.

Dans ce cas, l'ergonomie tente de proposer des méthodes de programmation afin que les programmes soient écrits de telle sorte que leur usage ultérieur en soit facilité.

b) Les environnements d'aide à la conception

A côté de l'amélioration de la présentation des programmes, l'ergonomie s'est également intéressée au processus même de la conception des programmes. Le problème est le suivant : comment aider efficacement l'informaticien dans sa tâche de conception, c'est-à-dire quelle assistance peut-on lui fournir pour élaborer des programmes ?

Un domaine de recherche ambitieux concerne la conception d'outils "intelligents" qui s'adaptent à la logique de pensée du programmeur en vue de lui fournir une assistance dans la conception et "l'accouchement" d'un programme (Werz, 1993). Par exemple, l'environnement *Smalltalk* est né à la faveur des recherches de Papert (1981) et Papert et Solomon (1986) *–sur le langage informatique pédagogique LOGO–* et des nombreuses découvertes du Xerox-PARC *–sur la mémoire sémantique, la manipulation directe et la planification de l'activité–* (Kay, 1993). La vocation de cet environnement est de permettre aux informaticiens de "*stocker, d'accéder et de manipuler de l'information de telles façons que ce système puisse évoluer à mesure que les idées du développeur progressent* " (Goldberg et Robson, 1983, p7).

Il existe également un autre pan de recherche sur la réutilisation des connaissances informatiques, et sur la manière de paramétrer les outils de programmation[18] pour accéder rapidement et facilement à ces savoirs. Ce courant porte sur l'analyse des processus de récupération de solutions provenant soit de source interne (mémoire du concepteur), soit de sources externes (supports techniques). Celles-ci sont évoquées par l'informaticien dans le but de formuler des réponses adaptées au problème rencontré (Burkhardt, 1997). Ces recherches, le plus souvent expérimentales, permettent d'apporter des éléments pour modéliser les composants réutilisables et faciliter leur accès et leur compréhension. On trouve des applications concrètes au niveau de la programmation orientée objet (Davies, Gilmore et Green, 1995) ainsi qu'au niveau des activités de supervision[19] (Houriez, Coupey, Visser, 1997).

c) L'apprentissage de la programmation

Cet axe de recherche porte sur l'étude à la fois des processus d'élaboration de connaissances de programmation, et des processus de transferts de connaissances lors de l'acquisition d'un second langage de programmation (Rosson et Gold, 1989). Dans le contexte d'évolution des langages informatiques (passage du paradigme procédural au paradigme orienté objet), ce type de recherche permet d'apprécier les effets positifs ou négatifs des transferts d'apprentissage (Détienne, 1990a, Chatel 1997). Brièvement, on regarde dans quelle mesure l'informaticien réutilise avec succès les connaissances de l'ancien langage pour développer dans le nouveau paradigme (effet positif). A l'inverse, on cherchera à savoir pourquoi il se réfère à des savoirs inappropriés qui le conduisent à formuler de mauvaises stratégies de développement dans le nouveau paradigme (effet négatif).

---

[18] Dans le passé, les outils de travail du programmeur se résumaient aux langages, à leurs compilateurs et à des éditeurs de texte très élémentaires. Maintenant, à ces outils s'adjoignent des aides très diverses : à l'analyse sémantique à la trace (exécution de programmes au cours de la construction), à la documentation (suivi et mise à jour des commentaires sur les programmes), à la preuve (de la conformité du programme aux spécifications du problème traité), à la navigation dans les librairies d'objet ou de codes ("Browser"), à l'élaboration des requêtes (pour l'extraction dans les bases de données), etc.

[19] Cette recherche effectuée sur l'activité supervision a comparé les représentations que les opérateurs se font des incidents de supervision aux représentations de ces incidents stockées dans une base informatique. L'analyse de la façon dont les opérateurs classifient les incidents dans cette base montre que les catégories choisies par les concepteurs de la base ne sont pas adaptées aux représentations que les opérateurs se font des incidents. Sur la base de ces observations, les auteurs ont formulé un certain nombre de recommandations pour un système informatique d'assistance à la réutilisation.

En somme, ces études essaient de comprendre l'organisation des connaissances de programmation et de déterminer les aspects de l'activité mentale qui dépendent ou non du paradigme du langage (Hoc, 1983 ; Batra et Antony, 1994).

d) Les outils d'assistance à la coopération dans la conception

Depuis peu, un domaine intéresse tout particulièrement les ergonomes : celui qui touche aux aspects collectifs de la programmation. Car, comme le fait remarqué Falzon (1995), la conception est rarement un processus individuel :

- d'une part, parce qu'elle met toujours en jeu deux acteurs : le concepteur et le demandeur. Ce demandeur intervient directement dans le processus de développement, par les besoins qu'il exprime, par les contraintes qu'il fixe, par les évaluations auxquelles il procède et par les corrections qu'il exige. Bref, il est un acteur indissociable du processus de conception ;
- d'autre part, parce que la conception informatique rassemble différents partenaires au sein d'une même équipe projet. Sa composition varie par la diversité des compétences représentées (techniques, fonctionnelles...), par la répartition et la distribution des tâches entre les acteurs (tâches d'analyse, de programmation ou de maintenance) et par une interdépendance entre les opérateurs (l'analyste devient le créateur des conditions du travail programmeur).

L'objectif de l'ergonomie est alors de rendre compte de la nature des interactions en jeu dans le développement, et de proposer des techniques d'assistance à la conception collective dans le cadre de tâches de "*concurrent engineering*" par exemple. On pourra citer les recherches effectuées par Darses et Falzon (1996) et D'Astous, Détienne, Robillard, Visser W. (1997) dont l'objectif est de construire un modèle cognitif des activités collectives mises en œuvre au cours du développement de logiciel. Leur méthodologie consiste à observer dans un environnement naturel et professionnel le comportement de concepteurs en interaction. L'objectif est de mettre en place une assistance logicielle à l'activité de synchronisation cognitive.

2.4 EN RÉSUMÉ

L'objectif poursuivi par l'ergonomie de la programmation est donc d'identifier, dans un premier temps, les caractéristiques des tâches et des raisonnements de conception, pour être capable, dans un second temps, de spécifier des environnements de développement qui soutiendront les activités de résolution et d'écriture de programme.

Elle s'intéresse aux pratiques individuelles de conception en tant qu'elles renvoient aux processus cognitifs qu'utilisent les concepteurs pour développer une solution. Mais, elle étudie aussi les pratiques collectives de conception car celles-ci mettent en évidences les processus de synchronisation des actions et d'allocation des tâches selon les compétences.

Les apports de ces différentes études, qu'elles soient expérimentales ou empiriques, sont importants. Grâce à elles, l'ergonomie de la programmation a acquis scientifiquement et pratiquement un savoir et un savoir-faire, qui, livrés sous une forme relativement consommable, ont contribué à améliorer les techniques de développement et la pédagogie de la programmation. Ces travaux ont essentiellement insisté sur deux aspects de l'intervention :

- tantôt sur l'application de théories de la cognition humaine à l'activité de conception. Ici, le rôle de l'ergonome est de définir des modèles du fonctionnement cognitif de l'homme engagé dans une activité de résolution de problème ;
- tantôt sur l'adaptation d'outils informatiques existants aux caractéristiques mentales du développeur et de sa tâche. Cette fois, le rôle de l'ergonome est de corriger et/ou d'ajuster un environnement de programmation en préconisant un ensemble de recommandations.

On peut d'ors et déjà dire que notre recherche s'inscrit dans ces deux cas de figure.

3. LES CARACTÉRISTIQUES COGNITIVES DE L'ACTIVITÉ DE CONCEPTION

Le travail des concepteurs est typiquement un travail mental, au sens d'une activité cognitive par excellence. Celle-ci a été étudiée par les psychologues comme un cas particulier de tâches de résolution de problème, en particulier des problèmes de conception.

3.1 UNE ACTIVITÉ DE RÉSOLUTION DE PROBLÈME

Les chercheurs opposent l'activité de conception à d'autres types d'activités sur la base des caractéristiques des problèmes traités et des raisonnements qui s'y manifestent (Hoc, 1987a ; Falzon, 1995). Aux problèmes d'induction de structure et de transformation d'états s'opposent les problèmes de conception.

a) Dans *les problèmes d'induction de structure d'abord,* l'opérateur doit identifier la structure de la relation entre un ensemble d'éléments donnés. Le diagnostic médical en est un exemple concret. A partir d'un ensemble désordonné de symptômes, identifié lors d'examens, le médecin tente de les organiser en un ensemble cohérent, c'est-à-dire de reconnaître une forme connue qui lui permettra de définir une pathologie particulière.

b) Les problèmes de transformation d'états impliquent quant à eux que l'opérateur trouve un "chemin", une méthode permettant de ramener une situation initiale insatisfaisante (ce qu'il connaît) à une situation cible désirée (qu'il connaît aussi). Pour cela, il dispose d'un certain nombre de moyens d'action (opérateurs de transformation) et doit respecter les contraintes sur les états intermédiaires (certains états sont interdits ou peu souhaitables) (Weil-Barais, 1991). De bons exemples sont la résolution de problèmes mathématiques, le jeu de la tour de Hanoï, ou bien encore, dans le cadre du travail, la régulation d'un dysfonctionnement diagnostiqué sur un processus technique.

c) Les *problèmes de conception* se caractérisent à l'inverse, par le fait que l'état initial (*le problème*) est mal défini, et l'état final (*la solution*) est à construire. La double tâche pour le concepteur consiste dès lors à définir le problème en même temps que sa solution. Autrement dit, la définition du problème se fait progressivement tout en le

résolvant. Par ailleurs, il peut y avoir plusieurs solutions admissibles à un problème : le produit final n'est qu'un possible parmi d'autres. Un expérimentation réalisée par Bisseret, Figeac-Letang et Falzon (1988) a mis en évidence que six concepteurs experts dans un même domaine produisaient des solutions différentes de celles des autres experts. Mais d'autres caractéristiques sont représentatifs de ce type de problème.

3.1.1 Spécificités des problèmes de conception

Un certain nombre de traits sont présentés comme spécifiques aux activités de conception (Bisseret, 1987 ; Darses et Falzon, 1996 ; Falzon, 1995 ; Visser et Hoc, 1990) :

- les problèmes tendent à être larges et complexes : non réductibles à des problèmes locaux ou à des sous-systèmes indépendants ;
- ils sont mal définis dans la mesure où des spécifications du problème manquent et où apporter une solution au problème consiste, en partie, à introduire des contraintes ;
- il existe de nombreux degrés de liberté dans l'état initial du problème ;
- la définition du problème et l'élaboration de la solution s'effectuent en interaction. Le problème ne pré-existe pas à la solution : l'un et l'autre sont construit simultanément
- il n'existe pas de chemin de solution prédéterminée de conception : à chaque fois il faut réinventer le chemin entre les spécifications et la production ;
- l'évaluation de la solution est reportée à l'établissement de la solution finale, ou en tout cas limitée, du fait que la génération de toutes les solutions possibles est coûteuse. De ce fait, les solutions finales sont satisfaisantes, et non pas optimales ;
- la résolution de problème exige la maîtrise et mise en oeuvre de multiples connaissances portant à la fois sur le domaine de l'utilisateur (connaissances fonctionnelles) et sur le domaine informatique (connaissances techniques). Ce qui entraîne souvent un développement collectif de la solution ;
- la formulation des spécifications et la réalisation de celles-ci sont distantes dans le temps ;

- enfin, bien que la personne puisse posséder des procédures quasi-automatiques, le raisonnement n'est pas procéduralisé. En d'autres termes, le raisonnement ne se déroule pas de manière automatique et irréfléchie mais s'ajuste au contexte.

En somme, pour tous ces auteurs, l'activité de conception repose donc sur une activité de résolution de problème qui nécessiterait à la fois la définition du problème (*"la construction d'un espace problème"*) et l'élaboration d'une stratégie de résolution originale et satisfaisante (*"Comment faire pour résoudre ce problème ?"*).
Cela dit, qu'en est-il d'une tâche familière, c'est-à-dire d'une situation peu originale, requérant une activité quasi-routinière sur des problèmes bien connus. S'agit-il toujours d'une activité de conception avec résolution de problème, ou bien, est-ce une tâche automatisée avec application de solutions prédéterminées ?
En fait, il est possible de classer la conception en fonction du degré de créativité qui se manifeste dans les produits de l'activité.

3.1.2 La conception : entre créativité et routine

Gero et Maher (1993, cités par Falzon, 1995) et Mayer (1989) distinguent ainsi deux grandes catégories de situation de conception. Il s'agit de :

- la *conception routinière* (*Routine* Design) aboutissant à des produits qui ne diffèrent pas des productions précédentes de façon significative. Il s'agit donc de situations dans lesquelles le concepteur adapte un schéma pré-établi de façon à le faire correspondre aux exigences du problème particulier.
- Les *situations de conception non routinières* qui, comme sa désignation l'indique, ne sont pas familières aux individus. Mais Gero et Maher affinent encore cette analyse en subdivisant en deux classes la conception. Pour chacune d'entre elles, les solutions adoptées diffèrent de façon "substantielle" des solutions construites dans le passé et nécessitent de transformer le schéma de conception habituel. On se trouve alors en situation d'apprentissage.
    * La première classe de conception non-routinière est celle de la conception innovante (*innovative design*). L'innovation provient

d'un choix de valeurs de variables qui s'étend au-delà des valeurs usuellement adoptées. C'est ce choix inhabituel qui, pour les auteurs, produit l'innovation, en élargissant le champ des possibles.

* La seconde classe est celle de la conception créative (*"creative design")* dont le caractère non routinier conduit les concepteurs à voir la réalité autrement, c'est-à-dire à changer de représentation et de schémas de résolution.

En somme, à la différence des précédentes études qui considéraient la conception uniquement sous l'angle de la résolution de problèmes nouveaux ; Gero et Maher, et Mayer définissent comme "activité de conception" aussi bien les tâches routinières que les tâches non routinières, lesquelles sollicitent respectivement des raisonnements automatisés ou originaux.

Dans le cadre de notre recherche, nous retiendrons cette dernière approche parce qu'elle est représentative des orientations actuelles de la conception. En effet, l'approche historique de la conception avait montré que les entreprise cherchaient à baisser les coûts informatiques, en rationalisant les pratiques de développement, et en recourant surtout à la réutilisation de composants de programmation. Dans ces conditions, l'activité de conception s'apparente de plus en plus à une tâche industrialisée, répétitive et donc de moins en moins créative.

Reste que l'on peut néanmoins déceler plusieurs faiblesses à ces différentes études.
En premier lieu, les auteurs s'intéressent à l'activité de conception, et aux concepteurs en général, sans pour autant discuter de la grande diversité des acteurs de la conception : les chefs de projet, l'utilisateur final, les développeurs, l'analyste, le maquetteur... Vraisemblablement, pour chacun de ces acteurs, les raisonnements, les contraintes, l'activité vont différer. Aussi, à trop restreindre, on risque de schématiser.
En second lieu, on peut se demander si les caractéristiques identifiées dans la conception peuvent s'appliquer à la totalité du processus de conception ou à certaines étapes seulement du projet de conception. On pourrait penser en effet que, selon que l'on en est aux étapes d'analyse ou bien de programmation, l'activité des concepteurs pourrait ne pas être de même nature, et donc plus ou moins familière. Enfin, ces études ont envisagé les raisonnements de conception uniquement sous l'angle de la

pensée humaine individuelle. Or, les théories développées par la "Sociologie du travail" (1994)[20] et par des auteurs comme Suchman (1987), Winograd et Florès (1989), Green, Davies et Gilmore (1996) sur la cognition située, soulignent le rôle fondateur du contexte qui peut être perçu comme un artefact qui structure les savoirs des opérateurs. En cela, la résolution informatique relèverait aussi des interactions entre l'informaticien et son environnement professionnel (Brangier et Bobillier Chaumon, 1996).

3.2 EN RÉSUMÉ

Pour la majorité des auteurs, l'activité de conception est par essence une situation de résolution de problème : non seulement l'état initial est mal défini (le problème), mais l'état final est à définir (ce sont les buts et la solution). Aussi concevoir, c'est donc trouver une solution en même temps que définir le problème. En outre, tout une série de critères distinguent les problèmes de conception de ceux rencontrés dans d'autres types de tâches (transformation d'états et induction de structure). Cela dit, on a vu aussi que des auteurs considéraient que des situations plus familières, et donc moins problématiques dans le cadre de la résolution, relevaient également du domaine de la conception. Se distinguait alors la conception routinière de la conception non routinière.

A présent, nous allons spécifier les processus cognitifs qui interviennent dans l'activité de résolution des problèmes de conception.

[20] Voir le périodique: Sociologie du Travail (1994) "*Travail et Cognition*", numéro spécial, n° 4/94.

4. LES ÉTAPES COGNITIVES DE L'ACTIVITÉ DE CONCEPTION

Que les informaticiens soient en situation de conception routinière ou non-routinière, l'activité de résolution qui s'opère recoure à des processus cognitifs que l'on doit identifier. D'après les études consultées, ces stratégies s'articulent idéalement autour de cinq grandes étapes cognitives. En fait, elles se chevauchent la plupart du temps. Il s'agit de :

1) la compréhension du problème,
2) la construction d'un plan de résolution ,
3) la réutilisation de solutions,
4) la mise en œuvre du plan de résolution,
5) l'implémentation de la solution retenue.

Nous allons revenir sur chacune de ces phases en indiquant leur fonction dans la conception informatique.

4.1 LA COMPRÉHENSION DU PROBLÈME

Pour trouver une solution, l'informaticien doit d'abord identifier le type de problème qu'il doit traiter. Pour cela, il use de deux processus que sont : a) *la représentation du problème*, b) *le diagnostic du problème.*

a) Se représenter le problème

En programmation informatique, ***la compréhension du problème*** est certainement la phase la plus délicate et la plus importante puisque le concepteur va tenter de traduire les requêtes vagues, inconsistantes et incomplètes des clients en un modèle interne moins vague, plus complet et représentatif des concepts qu'il a l'habitude de manipuler (Nguyen Xuan, 1987 ; Bisseret, Figeac-Letang, Falzon, 1988). La tâche de l'informaticien consiste à construire une représentation spécifique de la situation qui lui permettra d'élaborer ensuite la solution. Cette représentation a pour objectif de définir les données initiales, les données à rechercher, les buts à atteindre et les contraintes de la situation problème (Malhotra, Thomas, Caroll et Miller, 1980 ; Weil-Barais, 1991, 1994).

Fonctionnellement, l'élaboration de la représentation se déroule à partir de la généralisation d'un schéma de connaissances –récupéré en mémoire à long terme– aux caractéristiques de la situation : « [Cette représentation] *consiste à sélectionner un*

schéma et à remplacer les variables du schéma par les informations spécifiques fournies sur la situation » (Richard, 1990, p 97). Toutefois, cette représentation reste très labile car « *il suffit que la situation change, ou qu'un élément non remarqué de la situation soit pris en compte, alors qu'il ne l'était pas, pour qu'une représentation soit modifiée* » (Iosif, 1993, p 283).

Le support papier jouera un rôle très important durant cette étape, comme d'ailleurs durant tout le développement. Il fournit en effet un cadre concret pour l'exploration et la représentation graphique du problème sous des formes aussi diverses que du pseudo-code, des diagrammes, du texte ou des dessins (Bellamy, 1994). Il permet aussi de garder la trace des différentes versions de solutions dans la progression de la résolution, et de partager ses idées avec les collègues de travail pour mieux coopérer (Strüng, 1991).

b) Diagnostiquer le problème

D'une manière générale, le diagnostic relève d'un problème de classification dans lequel un objet ou un phénomène inconnu doit être identifié comme membre d'une classe d'objets ou de phénomènes déjà connus (Hoc et Amalberti, 1995).

Dans cette perspective, Adelson et Soloway (1985) et Rosson et Alpert (1988) ont montré que *la décomposition du problème* en sous-problèmes simplifiait le problème initial, en permettant de distinguer les parties pour lesquelles le développeur dispose de solutions préétablies et connues, de celles pour lesquelles il en est dépourvu. La décomposition dépend de l'expérience de l'informaticien. Dans le cas où il a déjà résolu des problèmes identiques, les connaissances acquises peuvent être utilisées pour diriger la décomposition (sources internes). Sinon, le système technique peut lui fournir des schémas de conception provenant de programmes conservés sur les bases informatiques (sources externes). En somme, l'enjeu de ce diagnostic est d'associer le problème à un espace problème connu, et donc déjà résolu, de l'expérience de l'informaticien (Boreham et Patrick, 1996).

On retiendra donc que "*comprendre le problème*" signifie que l'informaticien organise un ensemble d'éléments *a priori* hétérogènes et non significatifs, en une structure significative permettant le déclenchement d'une action, c'est-à-dire d'un processus de résolution.

4.2 LA CONSTRUCTION D'UNE SOLUTION

Une fois le problème et ses sous-problèmes caractérisés, il s'agit pour le développeur de construire un plan de résolution pertinent. Ceci implique différents processus cognitifs qui vont être détaillés.

4.2.1 L'identification et la gestion des contraintes

Bonnardel (1991) et Darses (1990) ont montré que la résolution d'un problème nécessite la satisfaction d'un ensemble de contraintes. Une contrainte représente *« une condition à satisfaire pour obtenir le but, elle est donc une description partielle d'un objet, et par conséquent, une description partielle de la solution »* (Darses, 1990, 10). Leur compréhension aide le concepteur à identifier les propriétés que l'application doit respecter. En fait, ces contraintes permettent de circonscrire l'espace des possibilités de conception en fournissant des jalons et des repères à l'activité (sous forme de buts et d'exigences) (Blandford et Barnard, 1995).

Lebahar (1996a) observe ainsi que la stratégie de l'architecte dans une tâche de conception architecturale est de ne pas introduire trop tôt des contraintes relatives à son projet qui le conduiraient à faire des choix prématurés dont le coût de la remise en question serait trop élevé. Lebahar suggère dès lors de comprendre l'activité de conception comme un processus de réduction progressive de cette incertitude qui pèse sur la représentation de l'objet. C'est l'intégration graduelle des contraintes adéquates qui conduirait à cette réduction.

Cette idée d'une dialectique s'effectuant entre la définition des multiples contraintes portant sur le problème d'une part, et leur gestion tout au long du problème d'autre part est reprise par Darses (1990) dans une étude expérimentale sur la conception informatique : la tâche consiste à développer une solution de réseaux informatiques suite à un appel d'offres. L'hypothèse est que la résolution du problème et la planification de l'activité reposent sur la définition et la gestion des contraintes. L'expérience a été réalisée sur six sujets qui se distinguaient par leur niveau d'expertise (expérimenté ou débutant) et leur type d'expertise (chercheur, programmeur, personne chargée de la maintenance...).
L'analyse des données a permis d'identifier deux types de contraintes guidant l'activité de conception : les contraintes de préférence que le concepteur estime

souhaitable de satisfaire (par exemple, *"le coût de la solution, sa robustesse")* et les contraintes de validité qui doivent être obligatoirement vérifiées par les objets de conception (par exemple, *"un câble "Thick" ne peut être le support d'un segment de réseau supérieur à 500 mètres"*).

Darses observe aussi que les contraintes de préférence sont plutôt utilisées par les experts alors que les secondes sont davantage employées par les novices. En outre, ces contraintes ne sont pas gérées de la même façon puisque les contraintes de préférence sont souvent formulées dès le début de la conception et satisfaites plus en aval du processus de résolution, alors que les contraintes de validité sont en général formulées et satisfaites simultanément.

Deux réserves nous poussent toutefois à relativiser la portée de ces résultats. D'abord, l'échantillon restreint utilisé pour cette expérience (six sujets) pose le problème de la généralisation de ses conclusions. Ensuite, l'expérimentation est une situation artificielle qui ne prend pas en compte les spécificités propres au contexte réel de travail. Celles-ci peuvent pourtant modifier les conditions de réalisation de la tâche. En effet, les pressions hiérarchiques, les contrôles incessants des clients, les relations avec les collègues de bureau... sont autant de variables qui affectent le comportement du concepteur et conditionnent le déroulement du processus de résolution.

Néanmoins, cette expérience fournit des indications très intéressantes sur la nature des stratégies impliquées dans la conception informatique, et confirme le rôle tout à fait prépondérant des contraintes dans la démarche de résolution de problème.

4.2.2 La mise en oeuvre de connaissances pour l'action : les schémas de connaissances

Un cadre théorique souvent rencontré pour rendre compte des connaissances utilisées dans l'activité de conception informatique est celui inspiré par la théorie des schémas. Un schéma peut se définir comme un ensemble organisé de connaissances stockées en mémoire qui représentent, sous forme générique, des concepts, des procédures, des événements ou des séquences d'événements (Détienne, 1989b). Ce modèle est surtout utilisé pour expliquer et décrire les mécanismes de compréhension des programmes. Ainsi, la compréhension de programmes se rapprocherait de la compréhension de texte en ce qu'elle consiste en partie à l'évocation de schémas relatifs au domaine concerné et en l'instanciation de ces schémas. Un schéma étant composé de variables, il sera

instancié quand les valeurs particulières sont liées aux différentes variables. Ce qui revient à dire que le sujet utilise les schémas stockés en mémoire pour comprendre un programme.

De nombreuses recherches, portant surtout sur le thème de la compréhension des programmes, se sont référées à ce cadre théorique (Détienne, 1986, 1988, 1990b); Soloway et Ehlich (1984).

Dans leur expérience, Soloway et Ehlich (1984) ont, par exemple, prédit que dans une tâche de complétion de programme (*le sujet doit compléter les parties manquantes ou corriger les parties inexactes d'un programme*), des informaticiens experts pourront inférer des lignes manquantes sur la base des schémas qu'ils possèdent en mémoire. En revanche, les sujets novices n'auront pas les connaissances requises pour faire ces inférences. Suite à la tâche expérimentale, les auteurs observent que la performance des experts est meilleure que celle des novices et que ces sujets compétents utilisent des schémas de programmation pour comprendre un programme. Sur la base de l'évocation de ces schémas partiellement instanciés, ils récupèrent en mémoire l'information manquante.
Dans la même expérience, ces auteurs ont créé une autre condition expérimentale dans laquelle la combinaison des schémas, dans les programmes utilisés, dérogeait à certaines règles du programmation. La prédiction est que les écarts de performance entre des experts et des novices ne sera pas significative dans une tâche de complétion. En effet, si d'un coté, le programme présenté ne correspond pas exactement aux schémas de résolution évoqués par l'expert ; de l'autre le sujet novice ne devrait pas être troublé par cette combinaison incorrecte. Les résultats obtenus confirment l'hypothèse : la performance des experts est proche de celle des novices. Comme le font remarquer Soloway et Ehrlich, ces résultats soulignent la fragilité de l'expertise. Les experts ont de forts systèmes d'attente sur la façon dont doit être construit un programme ; quand ces attentes ne sont pas satisfaites, leur performance se dégrade.

Dans ses recherches, Détienne (1986, 1988, 1990b) a étudié l'activité de compréhension de programmes par des experts. L'expérience consistait à présenter un programme de gestion écrit en Pascal. Les sujets prenaient connaissance de ce programme, instruction par instruction. A chaque instruction nouvellement proposée,

la sujet devait expliciter quelle information nouvelle était apportée et quelles hypothèses il pouvait élaborer sur le reste du programme. La consigne ne donnait aucune information aux sujets concernant le but rempli par ce programme. L'analyse a permis de déceler les mécanismes cognitifs mis en œuvre pour comprendre un programme, et en particulier ceux qui permettent de générer certaines inférences faites pendant la lecture. Ainsi, ces mécanismes sont des processus d'activation de schémas dirigés du bas vers le haut (reconnaissance) ou du haut vers le bas (sélection) et des mécanismes d'adaptation de schémas grâce à des indices évocateurs contenus dans le programme.

Détienne distingue également différents types de schémas qui interviennent dans la compréhension des programmes, parmi lesquels :

1. des *"schémas élémentaires"* qui représentent des connaissances sur la structure de contrôle et sur les variables du programme (domaine de programmation). Par exemple, un schéma de compteur représente les différentes connaissances que l'on a sur le mode d'utilisation d'une variable de type compteur dans un programme et les valeurs qu'elle peut prendre comme valeurs d'initialisation et de mise à jour ;
2. des "*schémas algorithmiques*" qui représentent des connaissances sur des catégories d'algorithmes, et qui sont constitués d'une combinaison de schémas élémentaires (domaine de programmation). Par exemple, un programmeur connaît des algorithmes de tri, de recherche... Ces algorithmes sont plus ou moins abstraits et dépendants d'un langage de programmation ;
3. des schémas relatifs au "*domaine d'application*" qui représentent des connaissances sur des types de problèmes particuliers. Par exemple, il sait, par expérience dans le domaine de la gestion, qu'un problème de gestion réalise, en général, des fonctions de type création, modification, suppression.

En fin de compte, quatre idées nous semblent intéressantes à retenir par rapport à notre objet de recherche :

1. d'abord, les connaissances de l'informaticien sont organisées sous la forme de schémas hiérarchisés qui représentent des séquences d'actions strréréotypées correspondant à des buts à obtenir. Ces schémas de connaissances seraient relatifs à un domaine de programmation (schémas élémentaires et algorithmiques) et à un domaine d'application ;
2. ensuite, ces schémas régulent les processus de représentation, de compréhension et de résolution des situations de conception ;
3. ces schémas peuvent également être réutilisés dans deux situations de conception différentes ;
4. enfin, l'organisation, la nature et la richesse de ces schémas différent entre le novice et l'expert. Ils peuvent d'ailleurs induire des erreurs lorsqu'ils conditionnent trop fortement les systèmes d'attente des experts.

L'importance de l'évocation des schémas dans l'activité de programmation semble donc acquise. Cependant, il faut noter que la plupart des travaux issus de ce courant se sont surtout consacrés à comprendre les activités de compréhension, et moins à celles de conception. Ce n'est que depuis les années 90, notamment avec l'arrivée des langages orientés objets à fort potentiel de réutilibilité, qu'on assiste à l'émergence d'un courant de recherche où la récupération de schémas est vue comme un processus central de la conception. Dans cette perspective, le schéma est perçu comme un guide qui pilote le processus de décomposition d'un problème et détermine l'ordre dans lequel les actions doivent être accomplies. Évoqués et utilisés à tout moment du développement, ces schémas permettent d'inférer des structures de données et des fonctions afin d'exécuter et de résoudre des parties du problème (Darses et Falzon, 1996).

Une illustration de ce mode de fonctionnement est les stratégies descendantes (top-down) et ascendantes (Bottom-up) qui correspondent aux différentes stratégies que peut utiliser l'informaticien pour résoudre son problème. Nous allons les décrire dans la partie qui suit.

4.2.3 La définition et l'utilisation de stratégies de conception.

Il s'agit de mettre en oeuvre un ensemble de patterns cognitifs susceptibles de générer des états intermédiaires menant de l'état initial à l'état final, respectivement des données du problème à une ébauche de solution (Adelson et Soloway, 1985). Différentes stratégies pourront contribuer à obtenir un début de résultat, parmi lesquelles on peut distinguer :

a) des méthodes par heuristique qui l'on associera aux stratégies descendantes ;
b) les méthodes ascendantes ;
c) l'identification de critères de conception ;
d) les stratégies par essai/erreur et fin/moyen

a) Les méthodes par heuristiques, plus générales et moins systématiques que celles par algorithmes, peuvent impliquer des règles telles que "*décomposer le problème jusqu'à ce qu'il soit constitué du plus petit problème pour lequel il existe une solution connue*" (Guidon, 1990b). Cette stratégie de résolution repose sur la démarche descendante (Top-Down), encore appelée "dirigée par les concepts" : l'idée est que concevoir et produire un objet, un programme, c'est partir de la définition d'un objectif général que l'on décompose en sous-objectifs moins abstraits eux-mêmes décomposés en sous-objectifs jusqu'à prévoir et réaliser la suite d'actions élémentaires qui aboutira au produit souhaité.

Cette approche descendante de la résolution a été constatée par Lebahar (1992) qui a montré que la première représentation d'un problème revêt souvent la forme d'une conjonction de sous-buts, exprimés comme des fonctions spécifiques qui sont traitées séparément par le concepteur-architecte. Non seulement l'architecte traite habituellement le but général "*maison à construire*" en le décomposant en ces sous-buts, mais ceux-ci sont généralement regroupés en classes avec des ordres de priorité exprimés sur celles-ci. Par exemple, décomposer le but "*construire une maison*" en sous-buts "*plan au sol et structure*", combinés entre eux conjonctivement (réaliser le sous-but 1 et le sous-but 2 ; par exemple : *déterminer la structure au sol* tout en *déterminant la couverture*), ou de façon pré-conditionnelle (réaliser le sous-but 1 puis le sous-but 2 ; par exemple : *prendre une décision sur la surface au sol de la maison*, puis *répartir la surface de chaque pièce d'habitation*) (Darses, 1990, p 4).

Dans le domaine de la conception informatique, Rist (1991) a analysé les stratégies de développeurs expérimentés en programmation procédurale. Il a montré que lorsque l'informaticien connaît bien toutes les exigences de la conception, alors les plans sont développés de manière anticipée et descendante (*i.e.* depuis la première à la dernière action exécutable). En fait, l'expérience que l'informaticien a d'une situation de développement le conduit à traiter préférentiellement certains stimuli et à rappeler plus facilement des schémas de résolution admissibles. En raison de cette connaissance préalable, on peut dès lors supposer que la stratégie descendante correspondrait davantage au mode de fonctionnement des experts qu'à celui des débutants. Les travaux de Sebillote et Bisseret (1986) confirment cette hypothèse. Ils révèlent que les concepteurs adoptent une démarche descendante dans les phases de conception pour lesquelles ils peuvent exprimer une expertise confirmée. Dans ce cas, l'adoption de la démarche descendante rendrait compte en fait d'une mise en œuvre de "modules" de connaissances acquis par l'expérience et intervenant à certaines phases de son activité de planification.

b) Une autre stratégie correspond à la démarche ascendante (bottom-up) ou "dirigée par les données" : le concepteur met en œuvre cette stratégie dans les cas où il s'appuie sur des points de détail du problème posé pour évoquer ou pour élaborer des schémas (Hoc, 1987a, Détienne, 1989b). Des indices sont ainsi extraits du problème à traiter et permettent de déclencher des schémas qui constituent des plans guidant le processus de conception. En fait, les étapes élémentaires du processus sont déclenchées par des informations que la personne remarque sur l'état actuel de son problème, et qui lui suggèrent des décisions possibles, à quelque niveau que ce soit par rapport à une hiérarchie d'abstraction des buts et sous-buts pour l'ensemble de la tâche.

Ainsi, Lebahar (1996b) note que les architectes dont il a observé la démarche de conception répètent très souvent que, pour la production de leurs toutes premières esquisses, ils cherchent à trouver "*au moins un bout de problème*". Cette stratégie ascendante est représentative des démarches de compréhension/correction de programme dont nous avions déjà parlé plus haut.

c) Une troisième stratégie consiste à définir une série de critères pour guider la recherche à travers l'espace problème et définir des buts de conception : par exemple, "*le programme doit être hautement efficient*" sera le moyen d'éliminer les solutions de l'espace problème les moins pertinentes.
L'expérience menée par Bonnardel (1991) illustre cette stratégie. Elle a cherché à caractériser l'activité d'évaluation et de sélection de solutions dans un contexte de conception de produits aérospatiaux. Différentes types de problèmes de conception (traditionnels et nouveaux) sont proposés à des concepteurs qui se distinguent par leur niveau d'expertise (expert Vs novice) et par leur type d'expertise (spécialisé sur des domaines précis de chacun des problèmes présentés). Bonnardel leur demande d'évaluer les solutions associées à chacun de ces problèmes (solutions physiques et solutions conceptuelles) et de sélectionner celle(s) qu'ils jugent préférables en justifiant leur choix. L'analyse des données montre que le tri des solutions s'effectue sur la base de critères de référence (par exemple : "*coût et fiabilité de la solution*"). Mais le chercheur observe aussi que le nombre de ces critères varient en fonction du niveau d'expertise (expert/novice) et du type de problème à traiter : si les experts expriment davantage de critères que les débutants pour la résolution des problèmes nouveaux, il n'y a en revanche pas de différence significative pour la résolution traditionnelle. En fait, l'expert se servirait de son expérience pour inférer des attributs évaluatifs à appliquer pour les problèmes originaux.

On notera donc que le concepteur emploie son expertise à formuler, propager et satisfaire des critères et des contraintes liés au problème, compte tenu de ses expériences passées. Cette stratégie est directement liée aux savoir-faire des concepteurs, et donc à l'expérience qu'ils possèdent.

d) Enfin, l'informaticien peut encore faire appel à deux autres stratégies pour résoudre ses problèmes : il s'agit d'une démarche de type *essai-erreur*, et de type *fin-moyen* (Hoc, 1987a). L'intérêt principal de ces stratégies est qu'elles permettent de construire la solution de manière itérative, et donc de procéder à des corrections ou à des investigations plus approfondies sur celles-ci.

1. La première, par *essai-erreur,* consiste a explorer des chemins de résolution pour tenter d'atteindre le but. Quand un chemin s'avère infructueux, on revient à un état antérieur pour explorer un autre chemin, et ainsi de suite. Cette stratégie nécessite que l'informaticien ait défini, au préalable, plusieurs solutions admissibles parmi son répertoire de réponses.

   La profondeur des explorations peut être variable : on peut ainsi privilégier une stratégie d'exploration en "*largeur d'abord*" (Breadth-first) ou en "*profondeur d'abord*" (deep-first). Dans le premier cas, "*en largeur d'abord*", on explore simultanément plusieurs chemins possibles jusqu'à une profondeur limitée. Puis, on retient les solutions qui paraissent les plus efficientes et qui seront examinées plus avant. Ce type d'exploration est surtout employé par les experts. Dans le second cas, "*en profondeur d'abord*", on explore un chemin jusqu'au bout. Cette dernière stratégie est plutôt caractéristique des novices.

   Afin de développer un modèle de conception de logiciel, Adelson et Soloway (1988) ont voulu observé le processus de résolution informatique par des concepteurs engagés dans le développement d'un système de messagerie électronique (SME). Les auteurs ont demandé aux développeurs de résoudre un problème qui, bien que présentant certaines similarités avec les problèmes qu'ils avaient l'habitude de gérer, pouvait tout de même être considéré comme nouveau dans la mesure où aucun sujet ne disposait d'une solution directement applicable. Adelson et Soloway observent alors que les trois experts étudiés exécutent une stratégie de résolution en "largeur d'abord" (« *breadth-first strategy* ») : ils vont comparer simultanément plusieurs solutions extraient de leur expérience afin de retenir celle qui leur paraîtra finalement la meilleure selon des critères prédéfinis.

2. La seconde stratégie, de type *fin-moyen,* se rencontre dans les situations de résolution où, à tout moment, le sujet a la possibilité d'évaluer les écarts entre l'état actuel et l'état final souhaité, et de tirer parti de tels écarts pour choisir les opérateurs les mieux à même de les réduire. La solution élaborée ou récupérée s'ajuste ainsi. Le modèle le plus représentatif de cette stratégie reste le modèle GPS (General Problem Solving) de Newell, Shaw et Simon (1959)

e) *En résumé*

Cette partie nous a conduit à présenter plusieurs types de stratégies qui permettaient au concepteur de déterminer au moins une solution admissible au problème rencontré. On retiendra que plusieurs facteurs pouvaient déclencher ces stratégies de résolution :

- *soit des facteurs externes* : nature des problèmes rencontrés (familiers ou nouveaux), nature des contraintes et des critères associées,
- *soit des facteurs internes* : liés à l'expertise du sujet dans le domaine informatique et à la disponibilité des schémas de programmation en mémoire.

Dans toutes ces stratégies, les références à l'expérience passée, aux schémas antérieurs se révèlent tout à fait prépondérantes dans la résolution. C'est pourquoi, il est nécessaire de décrire le processus de réutilisation en œuvre dans la conception informatique.

4.3 LA RÉUTILISATION DE SOLUTIONS

Il s'agit en fait de réutiliser des solutions développées lors du traitement antérieur d'un problème qui sont disponibles soit sur des sources internes (mémoire de l'informaticien), soit sur des sources externes (bibliothèques ou librairies d'entités informatiques).

Pour Visser (1995b), la réutilisation des connaissances se distingue de la théorie des schémas car les schémas représentent des connaissances générales et abstraites, indépendantes du contexte : par exemple, on peut récupérer un schéma très général de programmation que l'on peut instancier dans la plupart des contextes. Tandis que les connaissances réutilisées s'apparentent davantage à des savoirs circonstanciels, avec tout leur contexte particulier, leur trait de surface, leur particularité historique....

Plusieurs expériences ont tenté de mettre en évidence la fonction de la réutilisation dans la résolution informatique.

Ainsi, dans l'étude de Kamath et Smith (1992) (rapportée par Burkhardt et Détienne, 1995), ces auteurs observent que la programmation avec le langage C++ dans un projet traditionnellement réalisé en langage C conduit à la réutilisation de 136 des 227 classes proposées. Dans ce cas, l'élaboration d'un projet de résolution est donc

grandement facilitée par la récupération de solutions proposées par des sources externes.

Il peut arriver aussi que plusieurs solutions récupérées apportent, chacune, des éléments de réponse au problème posé, et qu'il faille alors les assembler pour développer une solution synthétique admissible. L'expérience de Bellamy (1992) indique que la tâche de programmation peut se caractériser par une recherche de différents fragments de programmes qui, une fois recombinés ensemble, vont former un programme approprié. Ces bouts de codes sont extraits de la mémoire de l'informaticien et des bibliothèques de programmes des environnements informatiques de Smalltalk/V[21].

Enfin une solution réutilisée est rarement d'emblée appropriée au contexte du problème. Toute une série de modifications sont nécessaires pour l'ajuster au problème. Pour Détienne (1991a et 1991b), les solutions développées représentent ainsi une instanciaton de solutions antérieures : la solution *source* (récupérée dans un certain langage) et la solution *cible* (qui est recherchée pour un autre langage) sont différentes instanciations d'un schéma de même niveau. D'autres solutions dîtes "*alternatives*" existent aussi : jugées peu convenables d'après les critères évaluatifs de l'informaticien, elles vont pourtant être choisies pour certaines de leurs instances qui apportent des pistes de résolution intéressantes.

Dans ce cadre, les processus concernent plutôt la récupération d'éléments antérieurs au problème à traiter et portent sur l'approfondissement de la relation entre solution source et cible afin de développer la nouvelle solution. C'est ce que Détienne nomme une situation de "*Old code reuse*" (Détienne, 1991b).

Burkhardt et Détienne (1995) ont quant à eux essayé de déterminer les mécanismes cognitifs et les représentations mentales intervenant dans la réutilisation. Sept experts en conception orientée objet ont participé à leur expérience. La tâche expérimentale consistait en l'alternance de phases de conception (résoudre un problème de gestion) et de phases de réutilisation (déterminer quels éléments de solution ils souhaitaient récupérer dans une librairie de composants). Leurs conclusions indiquent que la récupération de solutions repose sur la construction de règles de sélection permettant

[21] Environnement de développement informatique

de choisir entre plusieurs solutions stockées en mémoire et réalisant potentiellement un but : par exemple, l'une de ces règles considère "*le potentiel de réutilisation offert par la solution*", ou encore "*la difficulté que l'expert attribue à son développement*". En d'autres termes, le coût probable de la réutilisation d'une solution devient un élément prépondérant qui va conditionner le choix de celle-ci. Concernant les représentations mentales, les auteurs montrent que le recours à une représentation dynamique de la procédure (simulation, animation d'un modèle mental) est plus importante dans les phases de conception que dans les phases de réutilisation. Ces représentations permettent en effet l'évaluation d'une solution partielle tout en favorisant les interactions qu'elle est susceptible d'avoir avec d'autres sous-parties de solution.

Plusieurs limites doivent cependant être relevées dans cette recherche : d'abord, le faible nombre de sujets est un handicap certain pour la généralisation des résultats, même si, comme le souligne Burkhardt et Détienne, leurs conclusions correspondent à « *un constat récurrent sur les études portant sur l'activité des experts* » (p 95). Outre cela, la procédure expérimentale séparant phases de conception et phases de réutilisation nous semble tout à fait arbitraire. En effet, ces deux phases sont étroitement imbriquées pour la production d'une solution. Enfin, cette expérience s'est centrée exclusivement sur les processus cognitifs impliqués dans la réutilisation de solutions issues de sources externes. Il aurait été intéressant de les comparer avec ceux déployés pour la récupération d'éléments de sources internes.

Néanmoins, les prolongements empiriques de cette étude sont très intéressants puisque ces deux chercheurs ont proposé toute une série de recommandations destinées à améliorer les dispositifs d'assistance à la réutilisation des composants informatiques. Parmi celles-ci, on pourra citer l'idée de proposer une classification des situations de réutilisation (classées par "*buts de l'opérateur*" ou "*type de comportement recherché à travers le composant réutilisé*") qui cible précisément les activités qu'il sera possible ou nécessaire d'assister au cours de la conception avec réutilisation. La plupart des dispositifs actuels proposent en effet par défaut tout une collection de composants réutilisables, sans discrimination, ni classement. C'est à l'informaticien de faire le bon choix en opérant le plus souvent par tâtonnement.

Dans le même esprit de recherche appliquée, Perron (1995) a analysé l'activité de superviseurs de réseaux téléphoniques en identifiant leurs stratégies de réutilisation ainsi que les cas d'incidents typiques auxquels ils se référaient pour diagnostiquer et maintenir le système. Deux résultats de son étude nous paraissent importants à retenir : d'abord la description de l'incident passe par la définition de son contexte technique, organisationnel, et temporel. Ensuite, l'opérateur définit de multiples critères pour identifier cet incident et l'associer à son corollaire passé. A partir de ses conclusion, Perron proposé un dispositif d'aide technique à la réutilisation d'incidents typiques pour favoriser la tâche de supervision.

Les recherches que nous venons d'exposer jusqu'à présent se sont souvent bornées à l'étude de la réutilisation rétrospective, c'est-à-dire de la reprise de solutions développées dans le passé. Or, il se manifeste aussi des processus de réutilisation prospective par lesquels on conçoit pour pouvoir réutiliser un jour (ou parfois immédiatement). C'est ce que Détienne (1991b) appelle la situation de "*New code reuse*" (par opposition à celle de "*Old code reuse*" basée sur la réutilisation antérieure). Elle observe ainsi que lorsqu'un concepteur élabore une solution et la réutilise explicitement pour solutionner d'autres sous-problèmes, le développement de cette solution est alors influencé par l'anticipation des autres solutions qui en seront dérivées. On pourra ainsi citer l'étude d'Adelson et Soloway (1988) qui, dans leur expérience sur la conception d'un système de messagerie électronique, remarquent que les concepteurs anticipent des éléments à intégrer à un niveau de but élevé à partir des résultats des traitements effectués à un niveau moindre. Lorsque ces experts considèrent ces données comme primordiales, ils sont alors capables de les maintenir en mémoire et de les récupérer à un moment plus opportun. Sinon, ils vont noter ce qu'ils devront se rappeler : des contraintes à intégrer à la démarche de résolution, des solutions partielles à tester, ou encore des incohérences potentielles à vérifier. En somme, l'informaticien anticipe des connaissances futures probables à partir des résultats d'action en cours.

Finalement, ces différentes études ont pointé sur le rôle tout à fait primordial de la réutilisation dans la conception informatique : elle favorise l'élaboration des solutions face à de nouveaux problèmes par application, adaptation ou agrégation d'anciennes structures de connaissances. Mais la réutilisation revêt un autre intérêt pour notre

étude dans la mesure où elle est souvent présentée comme étant à la base de l'apprentissage et de l'acquisition des connaissances (Michel, 1995). Plus généralement, l'affectation d'anciennes structures de connaissances à de nouvelles situations de conception fait intervenir le raisonnement analogique, et renvoie directement aux problèmes des transferts d'apprentissage d'un contexte de travail vers un autre. C'est donc à ce titre qu'il sera intéressant de décrire les principes, les fonctions et les mécanismes du raisonnement analogique. Mais ceci fera l'objet d'une autre partie qui portera plus spécifiquement sur ce type de raisonnement. Pour l'instant, nous allons poursuivre la description des processus intervenant dans la résolution de problème en nous intéressant cette fois aux modes de planification de l'activité de conception.

4.4 LES MODES DE PLANIFICATION

Généralement, l'activité de conception est décrite en tant que démarche de planification hiérarchique mais aussi opportuniste. Dans un premier temps, nous montrerons que l'activité de conception a été considérée en tant que processus ***de planification hiérarchique***, associée à une démarche Top-down et Bottom-up. Dans un second temps, nous verrons que plusieurs auteurs ont insisté sur le caractère ***opportuniste*** de la conception. Des planifications antérieures peuvent ainsi être remises en cause et leur hiérarchie abandonnée, laissant la place à de nombreuses itérations sur le plan.

4.4.1 Une planification hiérarchisée

L'activité de conception est envisagée à travers l'élaboration d'un plan qui organise et planifie les étapes de résolution. Ce plan se décompose en une hiérarchie de buts et de sous-buts qui définit une suite de transformations allant du général au particulier ou inversement, et qui précise les actions à entreprendre à chaque étape (Adelson et Soloway, 1985 ; Hoc 1987a et c, 1988 ; Sacerdoti, 1974).

Hoc (1987a) a distingué deux directions possibles dans la planification du plan de résolution (1987a, p13), « *tantôt la représentation schématique ou (et) hiérarchisée d'un objet (...) sera plutôt prospective en construisant les étapes à parcourir à partir de l'état initial, tantôt cette planification sera plutôt rétrospective en procédant à partir des buts* ». La première correspond à la stratégie par hiérarchisation successive (ou Bottom-Up) utilisée préférentiellement par les experts ; la seconde représente

l'approche par raffinements successifs (ou Top-Down) employée davantage par les novices. Enfin, la planification hiérarchique joue un rôle important dans la simulation et le test des solutions : elle permet d'anticiper les résultats en projetant mentalement le plan (Lebahar, 1992).

En ce qui concerne les fondements de la planification hiérarchique, deux modèles s'opposent : *(i)* l'un conjoncturel désigne les méthodes de conception informatique comme principaux responsables de l'organisation structurée de l'activité (Traunmüler, 1988); *(ii)* l'autre structurel souligne le rôle formateur de l'expérience dans la composition des plans (Modèle de "Planification à partir de cas" ; Hammond, 1989).

- Pour Traunmüler (1988), l'organisation hiérarchisée de la résolution découle directement des méthodes de conception orientées "procédures" (par exemple, le cycle de vie traditionnel en "V") : on prescrit une succession d'étapes de conception que l'informaticien devra réaliser. Tout retour en arrière étant difficile, voire impossible, l'informaticien est alors obligé de planifier précisément toutes les actions qu'il souhaite mettre en œuvre en respectant très fidèlement le canevas tracé par la méthode de conception.

- Hammond (1989) pense au contraire que la conception des plans repose sur l'expérience de l'individu. Son modèle théorique de "*Planification à partir des cas*" suppose en effet qu'une personne se base davantage sur sa propre connaissance que sur des règles causales (*i.e.* les méthodes procédurales) pour créer et développer de nouveaux plans. En d'autres termes, comme pour le processus de réutilisation, une personne construit toujours de nouveaux plans à partir d'anciens. Il emmagasine ses expériences de planification dans la mémoire épisodique[22] et les organise autour de deux sortes d'indices : les buts à satisfaire et les échecs à éviter. Les premiers identifient les plans à récupérer selon les objectifs à atteindre ; les seconds correspondent à une sorte de signal d'alarme avertissant l'opérateur des risques de tel ou tel choix de plan de résolution.

[22] Selon Tulving et Thomson (1973), cette mémoire conserve les souvenirs d'événements et d'expériences personnels et professionnels

> En somme, la planification à partir de cas considère que les plans sont la résultante des expériences réussies ou ratées de l'individu. L'apprentissage s'y révèle crucial parce que l'individu doit constamment se référer à ses acquis afin de construire de nouveaux plans, tout en évitant de reproduire les erreurs du passé.

En définitive, toutes ces études tiennent pour acquis le fait que la conception informatique est une activité linéaire et figée, qui peut être planifiée à l'avance. Pourtant, ce point de vue strictement hiérarchique ne sous semble psychologiquement pas approprié pour rendre compte de la gestion des activités de conception, compte tenu des aléas inhérents au processus de résolution. C'est pourquoi, nous allons être amenés à discuter d'une nouvelle approche présentée tantôt comme rivale, tantôt comme complémentaire de la planification hiérarchique : il s'agit de la planification opportuniste.

4.4.2 Une planification opportuniste

D'autres chercheurs (Hayes-Roth et Hayes-Roth, 1979 ; Visser 1988, 1991, 1992b, 1995a ; Davies et Castell, 1991) ont donc estimé que l'approche hiérarchisée ne pouvait être en mesure de rendre compte de l'activité réelle des informaticiens, mais seulement de la tâche prescrite de leur travail.

Visser et Morais (1987, 1985) ont par exemple réalisé des expériences où ils demandaient à des ingénieurs de décrire leur activité de conception. Ces derniers la présentaient comme suivant un plan structuré hiérarchiquement. Ce plan ne correspondait cependant pas à l'activité réelle telle qu'elle fut observée par la suite. En effet, la tâche était organisée de façon opportuniste : l'informaticien allait dévier du plan pour suivre des chemins qui se révélaient plus intéressants, soit par leur coût du traitement cognitif, soit en raison de l'attrait technique et/ou fonctionnel qu'ils représentaient. En d'autres termes, lorsque des actions semblaient plus économiques à réaliser ou plus pertinentes à suivre, l'ingénieur abandonnait son plan initial pour se consacrer à la réalisation de cette procédure (Visser, 1995a).

De son côté, Suchman (1987) a défendu l'idée selon laquelle l'action est toujours située et contextualisée. Alors que de nombreuses recherches indiquent que l'organisation et la signification de l'action humaine dépendent de l'élaboration

mentale de plans, Suchman montre que le déroulement d'une action peut toujours être projeté et reconstruit dans les termes d'intentions précédentes et de situations typiques ; elle parle alors "*d'action située*". Dans cette acception, les plans et les représentations sont des produits de l'action située. La cohérence de l'action située est liée pour l'essentiel, non pas à des prédispositions individuelles ou à des règles conventionnelles, mais à des interactions locales contingentes des circonstances particulières où se trouve l'utilisateur. En d'autres termes, ce ne sont pas les plans qui guident l'action des individus, mais des interactions spécifiques de l'utilisateur avec son environnement de travail : après chaque intervention sur son environnement, l'individu en évalue les conséquences. Puis, il va sélectionner en fonction de ce nouveau contexte, la procédure d'action qui lui paraît la plus appropriée pour atteindre son objectif (Frohlich et Luff, 1989 ; Conein et Jacopin, 1994). En conséquence, les plans doivent être davantage compris comme des ressources pour l'action que comme des structures de contrôle de l'action.

Au final, il apparaît que les modèles opportunistes divergent d'une approche hiérarchique puisque les premiers reposent essentiellement sur l'usage d'une stratégie qui n'impose pas d'ordre de mise en œuvre des modules de résolution, alors que pour les seconds, cet ordre est déterminé à l'avance. Avoir une vision opportuniste de l'activité de conception, c'est en somme adopter l'idée que la production ou la modification des séquences détaillées s'opère par une communication ascendante ou descendante entre les niveaux d'abstraction différents. Ainsi, une décision, à un niveau donné d'abstraction, pourra affecter d'autres décisions à n'importe quel niveau, et on pourra intervenir dans l'espace de planification à tout moment.

Cela dit, l'opposition qui est faite entre planification hiérarchique et planification opportuniste ne nous paraît pas aussi définitive que cela. Il n'y aurait pas d'un côté, planification hiérarchisée, et de l'autre, planification opportuniste. Au contraire, l'activité de conception résulterait de l'articulation de ces deux conduites comme le suggère Jeffroy (1993, p16), au travers de son concept de "planification située" : *« la planification située recouvre la capacité de l'acteur à penser à l'avance le déroulement de ses actions tout en se laissant la possibilité de le remettre en cause à tout moment en fonction de l'évolution de la situation. Les plans qui sont ainsi élaborés ne sont donc pas des programmes qui, une fois définis, vont être déroulés.*

Cette prévision positionne dans le temps un certain nombre d'actions à réaliser et des événements attendus».

Le déclenchement de ces modes d'organisation de l'activité dépend de différents facteurs que nous allons à présent décrire.

4.4.3 Les facteurs responsables du mode d'organisation de l'activité

Des travaux ont cherché à identifier les facteurs responsables de l'organisation hiérarchique et/ou opportuniste de la conception. Ces résultats montrent que cette orientation dépend essentiellement de la structure du problème et des étapes du processus de développement, des découvertes de la conception et de l'expérience de l'informaticien.

Ainsi, l'étude de Guidon (1990a) indique que l'organisation hiérarchique de l'activité n'est possible que pour des problèmes bien structurés et clairement définis. Dans le cas contraire (problème mal défini), une gestion opportuniste de l'activité se révèle plus appropriée. L'auteur souligne aussi que la découverte de nouvelles exigences ou de solutions partielles déclenchera un mode de conception opportuniste.

Pour Visser (1992a), la spécification du problème s'appuierait sur une planification hiérarchisée de la tâche, avec des objectifs et des sous-buts clairement définis. Tandis que la phase de réalisation de l'application se déroulerait selon un mode opportuniste. Les conclusions de l'étude de Davies (1991) s'opposent à celles de Visser. En effet, pour lui, l'organisation structurée de l'activité relèverait plutôt des phases de programmation, alors que la gestion opportuniste caractériserait davantage les phases de maintenance et de correction du programme. En outre, Davies observe que cette conduite opportuniste est engendrée par les limitations de la mémoire de travail de l'informaticien ; ce qui en ferait un comportement typique des novices. En revanche, la planification hiérarchique caractériserait davantage la démarche des programmeurs expérimentés. L'étude de Rist (1990) confirme ces analyses : il a cherché dans quelle mesure l'organisation de l'activité de conception dépendait des connaissances dont disposait le concepteur. Il conclut que l'activité du concepteur est organisée hiérarchiquement si celui-ci connaît déjà la solution du problème auquel il est confronté (d'où le rôle important de l'expérience). S'il n'a pas de solutions admissibles, son activité sera alors gérée opportunément.

Malgré les divergences de certaines études, on note à nouveau que des facteurs internes (expérience) ou externes (étapes de la conception, découvertes durant le processus de résolution) à l'informaticien déterminent un mode de gestion particulier de l'activité (opportuniste Vs hiérarchisé).

En définitive, ces diverses approches ont essentiellement indiqué les formes prises par la conception informatique, en tant qu'elles visent à conduire un projet d'une analyse des besoins à la réalisation d'un programme. Sous cet angle, l'activité de conception peut être perçue comme la transformation par étapes d'un problème initial en une solution informatisée qui se construit dans le temps selon :

- un plan hiérarchique dans la planification hiérarchisée ;
- des actions contingentes de la situation problème dans la planification opportuniste ;
- une articulation de ces démarches dans la planification située.

C'est donc en fonction des circonstances et des découvertes de l'activité, du contexte de la conception, du niveau d'expertise mais aussi de la recherche du maximum d'effet pour le minimum d'effort (principe d'économie cognitive) que l'informaticien sera amené à abandonner son plan de conception pour procéder à des "actions situées".

Jusqu'à présent, nous avons vu comment l'informaticien comprenait son problème, construisait ou réutilisait la solution, planifiait la résolution. Il reste une étape décisive qui consiste à implémenter cette solution dans le système informatique, c'est-à-dire à transformer la solution conceptuelle en solution technique. C'est la phase de réalisation du programme.

4.5 L'IMPLÉMENTATION DE LA SOLUTION RETENUE

Après avoir caractérisé un plan de résolution susceptible de convenir au problème à traiter, il s'agit pour l'informaticien de le traduire dans le langage machine par un langage de programmation approprié. Cette étape de transformation s'appelle la phase de "*mapping*" ou de "*mise en correspondance*" des représentations homme-machine.
Cette "*mise en correspondance*" des représentations se déroule entre un domaine d'application (résolution du problème) et un domaine informatique (programmation de

la solution) (Détienne, 1989a). Le premier suppose un "modèle cognitif" du problème et de sa solution ; le second implique un "modèle informatique" de la solution (programmation). Il y a donc d'un côté les *représentations internes* qui se rapportent aux solutions énoncées dans les concepts propres à l'informaticien (domaine de la tâche ou de l'espace problème), et de l'autre, *les représentations externes* qui sont la traduction de ces solutions en programme informatique (Sholtz, 1991).
L'enjeu de l'implémentation consiste alors à passer d'une représentation à l'autre en perdant le moins d'informations possible ; c'est-à-dire de *« transformer une représentation interne qui a une structure donnée (input) en une représentation externe qui doit se conformer à une structure dépendante d'un langage particulier (output)»* (Détienne, 1991a, p 4). Mais plus la distance entre ces deux représentations est grande, plus la difficulté et les risques d'erreurs seront importants. Les auteurs parlent alors *d'incompatibilité cognitive* (Streitz, 1987, Senach, 1990), de *"Gap sémantique"* (Blackwell 1996, Tepenning et Summer, 1995), de "*gouffre*" entre l'évaluation et l'exécution (Théorie de l'action de Norman, 1984, 1991), ou encore de distances cognitives (Nanard, 1991) pour désigner l'espace conceptuel qui sépare le modèle cognitif de l'individu du modèle technique du dispositif utilisé.

Dans le cadre de notre recherche, nous allons revenir sur trois approches qui peuvent nous aider à comprendre les difficultés rencontrées par l'informaticien dans cette phase d'implémentation de solution, et plus généralement dans la collaboration homme-machine. Il s'agit des modèles de la compatibilité cognitive de Streiz (1987), de la théorie de l'action de Norman (1986) et des distances homme-machine de Nanard (1991).

a) Le modèle de la compatibilité cognitive de Streiz (1987)

Le modèle de Streiz donne un éclairage particulier au problème de l'implémentation de la solution. Il suppose que la facilité d'utilisation et de compréhension du système informatique est grandement améliorée lorsque cette application est compatible avec les modes de raisonnement de son utilisateur. En partant des enseignements de cet auteur, on peut conceptualiser la notion de compatibilité entre la solution externe de l'informaticien et le fonctionnement interne de l'environnement technique de la façon suivante :

- soit S(f) : le traitement par le Système technique d'une fonctionnalité, d'une opération, d'un algorithme "f" ;

- soit C(f) : le "modèle conceptuel" de la solution de développement "f" imaginé par le Concepteur (C) ;
- soit C(S(f)) : la représentation cognitive qu'a le Concepteur (C) de la réalisation de la fonctionnalité "f" par le système "S".

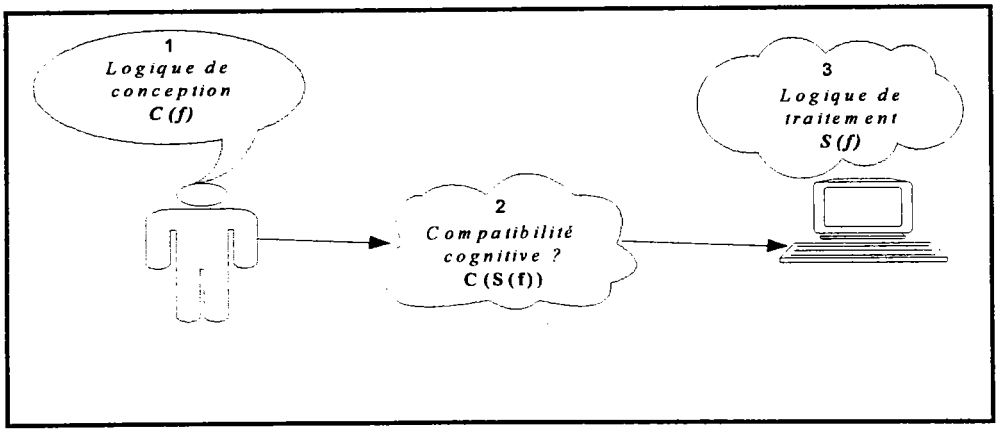


Figure 15 : Compatibilité cognitive

En conséquence, la "transférabilité" d'une solution conceptuelle en solution technique dépend en première instance de la compatibilité cognitive entre les deux entités du couple homme-machine. Plus un système sera compatible, moins la notion de désaccord entre les modèles en jeu sera marquée. En d'autres termes, moins il y aura de divergences entre S(f), C(f) et C(S(f)), plus le système sera orienté informaticien.

L'informaticien ne pourra donc implémenter convenablement une solution que si le paradigme technique dans lequel il va l'insérer est suffisamment proche de ses structures de connaissances. Sinon, il y aura incompatibilité cognitive, avec des risques d'erreurs de programmation et des difficultés de compréhension du paradigme.

Dans ces conditions, on peut imaginer que, dans le cadre des changements technologiques, les difficultés de transformation de la solution conceptuelle en solution technique résultent de deux phénomènes : soit l'expérience de l'informaticien est incompatible avec les caractéristiques du nouveau système technique, soit ce dispositif ignore les spécificités cognitives de ses utilisateurs.

b) La théorie de l'action (Norman, 1986)

Cette théorie part de l'idée selon laquelle la différence de représentation existant entre le monde physique réel et l'univers cognitif du sujet l'oblige à traduire le monde qui

l'entoure. Un utilisateur travaille avec des objectifs et des intentions. Il s'agit de variables psychologiques qui sont localisées dans le système cognitif de l'individu. Cependant, la tâche informatisée est toujours exécutée dans un système physique, selon les procédures opératoires qui résultent de l'interprétation que l'utilisateur fait des variables physiques du système en fonction des objectifs qu'il s'est fixé. Ceci revient à souligner l'importance de l'étape d'interprétation, qui met en rapport les variables physiques et psychologiques. De ce point de vue, la tâche est analysée selon un processus d'exécution et d'évaluation de l'action, et peut être traduit par sept étapes correspondant à l'activité de l'utilisateur (*élaboration d'un but, formation d'une intention, spécification d'une séquence d'actions, exécution des actions, perception de l'état du système, interprétation de l'état du système, évaluation de l'état du système par rapport au but qui était fixé*).

La théorie de l'action se propose dès lors de fournir un modèle pour saisir comment cette transition d'états est réalisée par l'opérateur. Elle a pour but d'analyser et de comprendre le processus de traduction qui sous-tend le comportement de l'opérateur. La théorie de l'action considère donc que les objectifs de l'utilisateur sont exprimés dans des termes qui sont pour lui pertinents, c'est-à-dire en des *termes psychologiques*, et que les états du système sont exprimés en ses termes propres, c'est-à-dire *des termes physiques*. Ceci amène Norman à distinguer l'état effectif de l'état perçu. L'état effectif est une fonction relative à des variables physiques formant le modèle conceptuel du système. L'état perçu correspond à la traduction de l'état effectif sous la forme d'une représentation mentale de l'utilisateur. Par conséquent, selon ce modèle, la divergence entre les variables psychologiques et physiques correspond au principal problème de la conception et de l'utilisation des systèmes interactifs. Norman représente cette divergence sous la forme de deux gouffres –gouffre de l'exécution et gouffre de l'évaluation–, franchis par deux ponts –pont de l'exécution et pont de l'évaluation–.

- Ainsi, l'écart entre l'objectif de l'utilisateur et le système physique est réduit par quatre segments qui constituent le pont de l'exécution. Il s'agit de la formation de l'intention, de la spécification de la séquence d'action, de l'exécution de l'action et enfin de l'interaction physique de l'utilisateur avec les dispositifs de saisie d'informations.
- L'évaluation, en revanche, implique de comparer son interprétation de l'état du système avec l'objectif qu'il s'est fixé. Un des problèmes est notamment de

déterminer dans quel état se trouve le système. Là aussi, l'écart entre le système physique et l'objectif de l'utilisateur est réduit par quatre segments qui représentent le pont de l'évaluation. Ce sont : l'affichage des données à l'écran, la perception et l'interprétation des données, et enfin, l'évaluation, c'est-à-dire la comparaison de l'interprétation de l'état du système avec les intentions initiales.

Ce modèle représente un réel intérêt pour notre recherche car il permet de considérer la phase d'implémentation comme une phase de traduction entre des variables psychologiques et des variables physiques. De surcroît, il désigne les processus susceptibles d'entraver l'implémentation de la solution. Les représentations mentales que l'informaticien se fait des principes de fonctionnement du système d'une part, et la "profondeur" des gouffres de l'exécution et de l'évaluation d'autre part, peuvent en effet conditionner la réussite de cette interprétation.

c) Modèle des distances homme-machine

Un dernier modèle, proposée par Nanard (1991), distingue trois types de distance homme-machine qui peuvent entraver la manipulation des outils informatiques. Il s'agit :

1. *d'une distance sémantique* : elle traduit l'effort mental que l'usager doit effectuer pour assurer la transformation de son problème (représenté mentalement dans les concepts de son monde) en termes des concepts proposés par le système ;
2. *d'une distance articulatoire* : elle représente l'effort mental que l'usager doit réaliser pour mettre en correspondance les concepts du système et les représentations qui leur sont associées. L'usager est ainsi conduit à remplacer une séquence d'actions par une seule opération conceptuellement simple ;
3. enfin, d'*une distance opératoire* : elle concerne l'effort psychomoteur que l'usager doit déployer pour exprimer une commande ou interpréter un résultat.

L'attrait de cette dernière approche réside dans le fait qu'elle met l'accent sur les origines possibles des difficultés d'interaction homme-machine, et indirectement sur celles liées à l'implémentation de la solution (par la distance sémantique).

Au total, l'implémentation d'une solution est donc une phase capitale de la conception car elle clôt le processus de résolution par la traduction des représentations conceptuelles en représentations techniques. Les différents modèles que nous avons présentés ont tous souligné que les difficultés rencontrées par les informaticiens étaient souvent la conséquence de l'inadéquation des dispositifs physiques avec leurs caractéristiques psychologiques. C'est pourquoi les conditions de mise en œuvre de la conception informatique reposent directement sur la qualité de l'interaction homme-machine. On peut d'ailleurs préciser que c'est dans cette problématique de rapprochement des représentations homme-machine que nous nous placerons ; l'enjeu étant d'offrir des environnements de travail qui optimisent non seulement l'activité de conception, mais qui favorisent aussi les transferts d'apprentissage.

4.6 EN RÉSUMÉ

Les processus mentaux mis en oeuvre pour comprendre et résoudre un problème informatique consistent en (Cf. Figure 16)

1) la compréhension du problème par l'élaboration d'une représentation pertinente et le diagnostic de la situation problème ;
2) l'identification des contraintes ;
3) le rappel de schémas de connaissances des sources internes/externes ;
4) La construction et/ou la réutilisation de stratégies de résolution admissibles ;
5) l'organisation du plan de résolution (hiérarchique / opportuniste) ;
6) son implémentation dans l'environnement informatique considéré (mise en correspondance entre le domaine du problème et le domaine informatique).

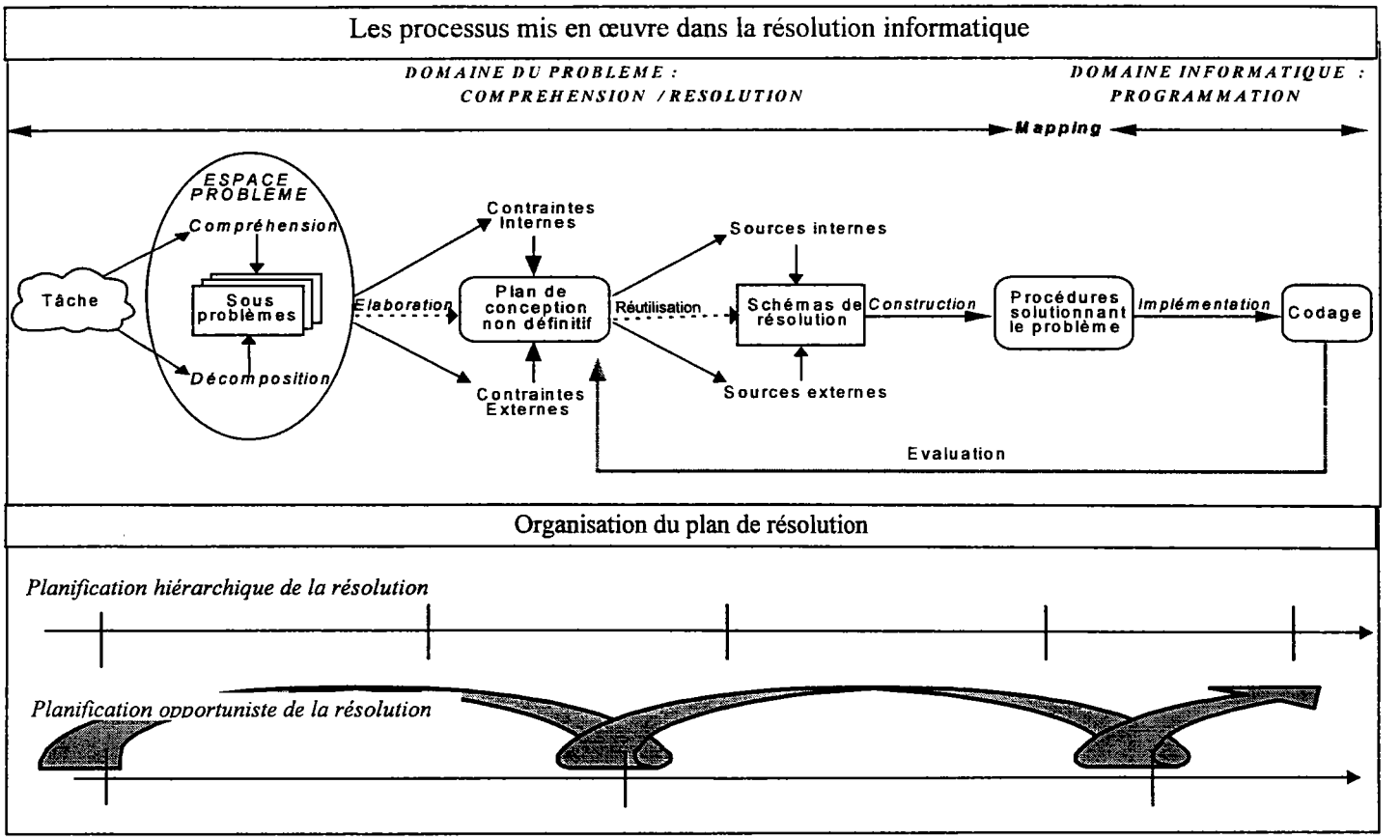


Figure 16

Les activités mentales mises en œuvre durant la conception informatique

5. LES TRANFERTS DE COMPÉTENCES DANS LA CONCEPTION

5.1 LE RAISONNEMENT ANALOGIQUE : DE QUOI S'AGIT-IL ?

Lors de la résolution d'un problème en informatique, le concepteur peut opérer un transfert d'une première situation appelée "**source**" vers la situation "**cible**" qu'il doit solutionner (Gentner, 1989). L'analogie peut être décrite comme une mise en correspondance entre des éléments et des descriptions d'un domaine source et d'un domaine cible. Elle est utile quand le sujet est familiarisé avec la source et peut apparier des éléments familiers ou des relations de la tâche source à des éléments ou des relations non familières ou inconnues de la tâche cible (Barcenilla, 1993).

On parlera dès lors de transferts d'apprentissage pour désigner l'utilisation de connaissances que l'on dispose sur une situation source bien connue et mémorisée, pour engendrer de nouvelles connaissances concernant la situation cible ; le transfert désigne cet emprunt de connaissances.

Les différentes études menées sur ce thème (Nguyen-Xuan 1990 ; Richard 1990; Perron, 1994 ; Fortin et Rousseau, 1994 ; Visser 1995b, 1995c) font ressortir un certain nombre de caractéristiques que nous estimons tout à fait prépondérantes par rapport à notre recherche. Il s'agit :

1. des *fonctions du raisonnement analogique* car le transfert analogique n'intervient pas seulement pour la résolution d'un problème de conception, il joue un rôle tout aussi important dans la compréhension des situations et dans l'apprentissage d'un nouveau dispositif (outil, langage...).
2. des *mécanismes de fonctionnement du raisonnement analogique*. Les modèles explicatifs que nous présenterons vont nous aider à identifier les origines des transferts négatifs (ou inappropriés) entre deux situations de conception.
3. des *transferts intra-domaine et les transferts inter-domaine* car ils montrent que le succès de la résolution dépend en grande partie du contexte d'où sont tirées les connaissances de référence;

4. *des types de connaissances transférées*. Cela revient à déterminer les éléments, les concepts, les schèmes d'actions et de résolution qui peuvent être transférés entre deux situation de conception.

5.2 LES DIFFÉRENTES FONCTIONS DU RAISONNEMENT ANALOGIQUE DANS LA CONCEPTION

Le raisonnement analogique peut avoir trois sortes de missions : il peut tantôt être utilisé pour comprendre une situation, tantôt servir à résoudre un problème, tantôt être exploité pour générer de l'apprentissage.

5.2.1 La compréhension

Comprendre par analogie signifie que l'on va emprunter un réseau de relations d'une situation source pour organiser un ensemble d'objets d'une situation cible (Nguyen-Xuan 1990). Cela correspond en fait à l'instanciation de schémas de connaissances qui a déjà été exposée dans la compréhension des programmes informatiques.

5.2.2 La résolution

Résoudre par analogie conduit à considérer un réseau de relations d'une situation source à partir duquel on effectue des inférences. Ces inférences, valides dans la situation source, sont ensuite transférées dans la situation cible. Selon Visser (1992a), plusieurs modes de recherche de solutions coexistent :

- la récupération de la solution telle quelle en mémoire, qui n'est pas à proprement parler une résolution de problème dans la mesure où le sujet dispose d'une procédure lui permettant d'atteindre son but ;
- l'application de connaissances relatives à une catégorie de problèmes, ce qui consiste pour le sujet à résoudre et à appliquer les connaissances relatives à cette catégorie en les particularisant pour les données du problème ;
- la récupération en mémoire d'un problème similaire quand le sujet ne dispose pas de catégories, mais qu'il connaît un problème similaire, ce qui peut lui permettre de résoudre son problème par un raisonnement analogique ;
- la construction d'une solution à partir de connaissances générales.

5.2.3 L'apprentissage

Si le recours à l'analogie s'avère réussi –et même s'il ne l'est pas–, les résultats sont alors stockés (sous la forme de concepts, de règles ou de schémas de problème) et constitueront une base pour de nouveaux appariements (Perron, 1994).

L'informaticien recourt à ce raisonnement analogique lorsqu'il doit se former à un second dispositif ou langage informatique. On parle alors de **réapprentissage.**
Détienne (1990a) a ainsi étudié des informaticiens qui, expérimentés dans un environnement procédural C, devaient se former à un nouvel environnement orienté objet (C++). Elle remarque, chez les sujets débutants, la préexistence de certaines pratiques procédurales dans l'utilisation du nouveau langage orienté objet (LOO). Cet apprentissage analogique conduit alors à deux effets : un effet positif lorsqu'il facilite et accélère l'apprentissage et l'utilisation du nouveau langage (par exemple, la syntaxe est identique entre les deux paradigmes) ; un effet négatif lorsque la persistance de certaines habitudes de programmation empêche la mise en œuvre des concepts OO. Le premier est appelé *transfert d'apprentissage positif*, le second *transfert d'apprentissage négatif.*
Mais il se peut aussi qu'une personne soit confrontée, pour la première fois de sa carrière, à un nouveau dispositif ; il doit alors développer des connaissances spécifiques pour ce système. Dans ce cas, il y a tout à apprendre, et on parlera alors **d'apprentissage.**

La formation à ce nouvel outil repose sur un apprentissage par l'action : le sujet a la possibilité d'observer le résultat de ses propres actions, et de modifier son comportement en fonction des objectifs qu'il s'est fixés. Selon Nguyen Xuan (1987), quatre mécanismes interviennent dans cet apprentissage particulier :

1) la construction de l'espace problème ;
2) la collecte des mauvaises actions et construction des procédures pour éviter les mauvaises actions ;
3) la collecte des bonnes actions et la construction des procédures de génération de sous-buts ;
4) la structuration de ces nouveaux sous-buts.

En somme, dans l'apprentissage par l'action, l'action paraît être plus une *occasion* de connaissances qu'une simple *application* de connaissances.

Détienne (1990a) a d'ailleurs mis en évidence ce mode de fonctionnement dans son étude sur l'apprentissage d'un langage OO. Les informaticiens débutants cherchaient souvent à agir, c'est-à-dire à écrire leur programme le plus rapidement possible. Puis, ils se basaient sur les informations données comme résultats de leurs actions (notamment les messages d'erreur générés à la compilation) pour apprendre à utiliser le système de programmation objet.

De même, Frese et Hesse (1995) ont montré que l'apprentissage des méthodes et des outils de programmation s'effectuait à 40% sur le "tas", c'est-à-dire au cours de la réalisation de l'activité. Mais leur enquête souligne aussi que lorsque les informaticiens disposent de connaissances théoriques par des formations générales à l'informatique, cela renforce d'autant plus leur capacité d'auto-apprentissage : « *ceux qui ont appris à apprendre sont nettement avantagés* » (p 9). Ce qui revient à dire que l'apprentissage par l'action peut être renforcé par des raisonnements analogiques effectués à partir d'anciennes connaissances.

5.2.4 En résumé

Au final, on retiendra donc que la fonction du raisonnement analogique consiste à mettre en correspondance une source et une cible, afin de produire une représentation (cas de la compréhension) et une solution (cas de la résolution) plausible qui est ensuite évaluée ou justifiée. Mais ce n'est pas sa seule vocation. En effet, l'analogie n'est pas un simple transfert accompagné de corrections locales. L'informaticien a la possibilité de faire une utilisation heuristique de l'analogie, en développant un apprentissage par l'action : la création par analogie d'une connaissance vraiment nouvelle est alors possible si on emprunte des connaissances à plusieurs domaines-source.

5.3 LES PRINCIPES DE FONCTIONNEMENT DU RAISONNEMENT ANALOGIQUE

Parmi les modèles qui décrivent les mécanismes de raisonnement analogique, trois approches nous semblent particulièrement puissantes pour aborder et expliquer les problèmes posés par les transferts d'apprentissage dans le cadre des changements techniques. Il s'agit de "*l'approche pragmatique*" d'Holyoak (1984), du principe "*d'encodage spécifique*" de Tulving et Thomson (1973) et du "*raisonnement à partir du cas*" décrit par Visser (1995 b et c).

5.3.1 Trois modèles explicatifs

a) L'approche pragmatique du raisonnement analogique d'Holyoak

Chez Holyoak (1984), dans l'analogie, chaque problème analogue est considéré comme une instance d'un schéma de problème plus général, lequel est organisé hiérarchiquement avec un état initial (but, ressources, contraintes) et un plan. La résolution du problème par analogie implique alors quatre étapes (Perron, 1994) :

1. des représentations mentales des problèmes sources et cibles doivent être construites ;
2. l'analogie potentielle doit être remarquée : certains aspects du problème-cible doivent servir d'indices de récupération pour que le sujet se souvienne du problème source ;
3. une mise en correspondance initiale et partielle doit être effectuée entre les éléments (objets et/ou leurs attributs et relations) des deux situations ; cette mise en correspondance peut être essayée à différents niveaux d'abstraction ;
4. pour finir, la mise en correspondance doit être étendue en récupérant et en construisant à partir de la source de nouvelles connaissances à propos du problème-cible.

De ce point de vue, le raisonnement analogique aurait un double effet : il permettrait d'une part, de créer de nouveaux schémas par induction et, d'autre part, d'appliquer des procédures en faisant des inférences sur la cible. Sa fonction serait alors de favoriser la compréhension de nouveaux problèmes et l'apprentissage.

b) L'encodage spécifique

Selon ce modèle, qui repose sur les travaux de Tulving et Thomson (1973), le raisonnement analogique articulerait trois processus : 1) d'abord, l'accès à la situation/solution source, 2) ensuite, la mise en œuvre d'un processus de récupération par génération d'indices de récupération, 3) enfin, la mise en correspondance de la solution récupérée aux conditions de la situation problème.

1. *Accès à une situation/solution source*

   Une fois construite la représentation du problème cible, la première étape de la réutilisation analogique concerne l'accès en mémoire à une ou plusieurs solutions sources et la sélection de l'une d'entre elles. Mais ce processus présuppose au préalable que des indices de récupération soient générés pour permettre la sélection d'un ensemble de solutions candidates parmi l'ensemble du répertoire stocké en mémoire (on en verra un exemple dans le paragraphe suivant).

2. *Processus de récupération par génération d'indices de récupération*

   La récupération est le processus par lequel l'information emmagasinée en mémoire à long terme (MLT) est réactivée en mémoire à court terme (MCT ou mémoire de travail). La réutilisation de cette connaissance va soutenir l'action en cours. La récupération se fait toujours en relation avec un "*indice de récupération*" cognitif (*retrivial cue*) qui spécifie l'information à rechercher dans "*la mémoire épisodique*" (Visser, 1995c). Par exemple, un indice de récupération pourrait être : "*rechercher le programme permettant d'afficher un objet graphique de couleur rouge*". Cet indice doit préciser le contexte de l'encodage et la nature de l'information recherchée en mémoire, par exemple, "*rechercher une fonction informatique développée dans un projet lambda* [contexte] *qui permet d'afficher une liste d'items sous forme hiérarchique* [nature de l'information]". La situation ancienne qui reçoit la somme d'activation la plus forte sera alors rappelée en tant que situation analogue.

   A ces indices de récupération s'ajoutent des "*critères d'évaluation*" qui permettent de décider si le processus de récupération doit ou non se poursuivre. Plus précisément, ils vont évaluer la validité de l'information

récupérée par rapport au contexte et aux objectifs de l'intervention, et prescrire, si nécessaire, des ajustements sur celle-ci (Cf. Figure 17)

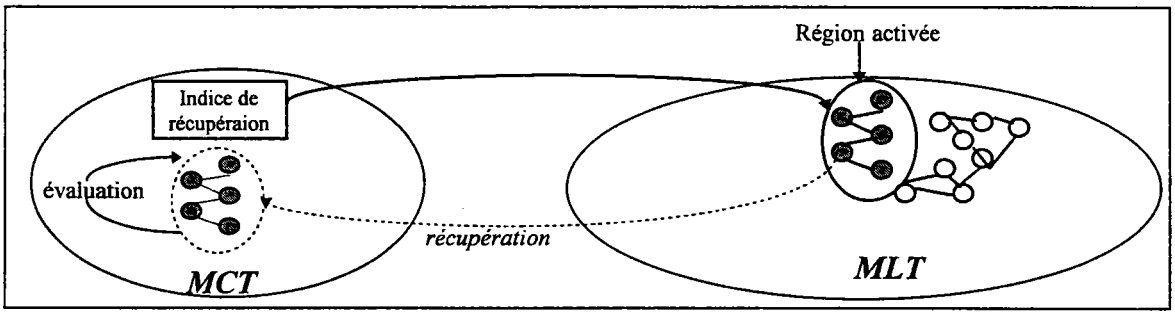


Figure 17
Processus de récupération analogique

3. *Mise en correspondance de la solution récupérée aux conditions de la situation problème (objectifs, contraintes...)*

   Ultime phase, la mise en correspondance vise non seulement à l'ajustement de la solution au contexte de conception, mais elle cherche aussi à développer de nouvelles connaissances.

Selon ce modèle, la fonction du raisonnement analogique serait de permettre la résolution de problème et l'apprentissage.

c) Le raisonnement à partir de cas

Cette approche née aux Etats-Unis dans les années 80 est souvent présentée comme une théorie sur la cognition humaine bien qu'elle soit massivement utilisée comme modèle pour la conception des systèmes experts.

Elle se base sur les éléments suivants (Visser 1995b, 1995c) :

- l'unité de base de connaissance est le cas : un cas est un terme qui renvoie à un épisode particulier relatif à une prise de décision particulière ou à la résolution d'un problème spécifique. L'expérience antérieure (d'un système, d'un opérateur) peut être représentée sous la forme d'un ensemble de cas ;
- l'expertise c'est l'expérience ;
- les experts raisonnent par analogie.

Pour ce dernier modèle, la fonction du raisonnement analogique serait essentiellement de fournir des cas pour la résolution du problème.

5.3.2 Analyse comparative de ces trois modèles

Le modèle de raisonnement à partir de cas (RPC) se démarque fondamentalement des deux précédents par le fait que la connaissance source, réutilisée par analogie, se trouve au même niveau d'abstraction que la cible (le problème à résoudre). En d'autres termes, dans le RPC, le cas récupéré est spécifique à une cible, alors que pour les deux autres approches, les connaissances rapatriées sont générales et abstraites (schémas de connaissances, procédures) et doivent par conséquent subir des adaptations.

En outre, les conditions de déclenchement de ces raisonnements diffèrent selon le niveau d'analogie entre les contextes source et cible : une analogie définie comme "profonde" entre deux situations –c'est-*à-dire une analogie dans laquelle les relations causales essentielles sont maintenues (analogie de buts par exemple)*– va favoriser un transfert analogique à partir de cas. Tandis qu'une analogie de "surface" –*similarité sur certains aspects de la demande ou du problème*– va plutôt être à l'origine d'un transfert par instanciation de schémas et récupération. On parlera alors plutôt d'évocation des schémas.

Enfin, ces trois modèles envisagent différemment le rôle que tient le raisonnement analogique dans la maturation des processus cognitifs : contribue-t-il davantage à la compréhension des situations, à la résolution et/ou à l'apprentissage ? le tableau ci-dessous résume ses différentes fonctions selon les approches considérées.

		Fonctions cognitives du raisonnement analogique		
		Compréhension	Résolution	Apprentissage
Modèles de raisonnement analogique	Modèle pragmatique	●		●
	Indices de récupération		●	●
	Raisonnement à partir de cas		●	

Figure 18
Tableau comparatif des modèles explicatifs du raisonnement analogique

Une critique générale peut toutefois être adressée à l'encontre de ces trois approches : toutes se basent sur des situations dans lesquelles les problèmes analogues sont fournis aux sujets par les expérimentateurs et non pas récupérés spontanément par les sujets eux mêmes. Elles introduisent donc une limitation par rapport à l'étape importante de récupération en mémoire. Par ailleurs, si les deux premiers modèles sont universels

dans le sens où ils peuvent décrire et expliquer les mécanismes de raisonnement des novices et des experts, le modèle RPC n'est valable que pour des personnes expérimentées, et ne peut donc rien dire sur le fonctionnement des novices. Ce qui limite fortement son application.

Malgré ces réserves, ces modèles permettent de repérer des facteurs susceptibles de conditionner l'efficacité du raisonnement analogique (transferts positifs ou négatifs) dans le cadre de la conception informatique. Il peut s'agir :

- de la fiabilité des processus servant à définir la cible et qui sont utilisés pour récupérer et adapter la source ;
- de la nature des connaissances détenues par l'informaticien qui peuvent être de type général et abstrait (schéma) ou spécifique (cas du RPC) ;
- du contexte de rappel qui détermine le type de connaissances récupérées.

A présent, nous allons déterminer quelles sont précisément les limites du raisonnement analogique.

5.3.3 Les limites du raisonnement analogique dans l'activité de conception

Le raisonnement analogique dans la conception informatique a pour but de réutiliser des solutions, des représentations, des connaissances antérieures pour résoudre, comprendre et/ou réaliser un apprentissage sur une nouvelle situation cible. Mais certaines fois, l'analogie peut constituer un frein à l'apprentissage et à la résolution, en raison :

1. du dysfonctionnement des mécanismes de récupération et d'évaluation qui conduit à transférer des séquences d'action du domaine source (où elles sont permises) vers le domaine cible (où elles sont illicites). Ce qui engendre des incidents et des erreurs ;
2. du contexte plus ou moins familier de la situation cible ;
3. d'une analogie trop parfaite qui empêche d'explorer d'autres manières de trouver des solutions ;
4. de la rigidité et de l'enracinement des apprentissages qui empêchent tous recours à de nouvelles connaissances ;
5. du recours à des biais cognitifs.

UNIVERSITE DE METZ BU Lettres S C D

1) *De l'effet des dysfonctionnements des mécanismes de récupération et d'évaluation des connaissances sur l'efficacité du raisonnement analogique*

Si l'on se réfère au modèle de l'indice de récupération, il s'avère que la performance du raisonnement analogique serait étroitement liée à l'adéquation du couple "*Qualité de l'indice de récupération / Prégnance des solutions en mémoire*" . Ainsi, plus la qualité de l'indice de récupération est élevée, plus grande sera la chance de retrouver la connaissance recherchée. De même, plus la trace mnémonique de l'information recherchée est forte, plus grande sera la probabilité de récupérer cette donnée. Par conséquent, les transferts d'apprentissage négatifs résulteraient de deux phénomènes complémentaires :

- d'une part, l'inexactitude ou l'incomplétude de l'indice de récupération conduirait à rapatrier une connaissance (source) non pertinente par rapport à la situation cible. Le mauvais diagnostic du problème pourrait être à l'origine d'une définition erronée de l'indice ;
- d'autre part, l'absence ou le manque de fiabilité des critères d'évaluation induirait un mauvais filtrage des solutions ; les bonnes connaissances n'étant pas différenciées des mauvaises.

2) Du rôle du contexte dans l'efficience du raisonnement analogique

Le contexte affecte également les conditions de réussite du raisonnement analogique. En particulier, dans le cas d'une situation de conception non familière, il est plus difficile d'identifier des catégories de problèmes analogues et de réutiliser les solutions qui leur sont associées. Pour éviter cet écueil, Falzon et Visser (1989) ont proposé un assistant informatique qui aide à décrire le problème cible et qui propose des catégories de solutions admissibles.

L'étude de Chatel, Détienne et Borne (1992) sur les transferts d'apprentissage entre un langages familier (procédural) et l'autre nouveau (orienté objet) prolonge cette idée. Elles démontrent que les structures de connaissances auront d'autant plus de chances d'être instanciées dans le nouveau langage que l'informaticien se trouve confronté au même type de problème. En d'autres termes, le contexte familier de la conception devient un artefact du transfert d'apprentissage : plus les problèmes se ressemblent d'une situation à l'autre, plus grande sera la probabilité que le développeur instancie les pratiques cognitives associées au langage source.

3) *Des conséquence d'une analogie "trop parfaite" sur le raisonnement analogique*

Michel (1995) rapporte les dangers d'une analogie trop parfaite sur les transferts d'apprentissage. Lorsqu'il existe (ou lorsque l'individu suppose à tort...) un isomorphisme entre le domaine source et le domaine cible *–en termes de buts, d'objets et de contraintes–*, le sujet peut rester prisonnier du domaine source. Cette situation risque de l'empêcher de considérer d'autres points de vue de la situation cible. A l'inverse, si le recouvrement est trop faible, un risque d'interprétation erronée à partir de données non pertinentes du domaine source est également possible.

Douglas (1983) (cité par Barcenilla, 1993, p 72) a analysé des protocoles verbaux de personnes qui, habituées à travailler avec une machine à écrire, apprennent un traitement de texte. Il ressort que ces sujets se forment au système informatique par analogie avec la machine à écrire, principalement parce qu'ils présupposent une similarité du clavier, de l'écran du micro avec la page tapée, et des tâches dans l'édition et la frappe. Selon Douglas, la difficulté principale lors du transfert analogique est que les connaissances sur les opérateurs sémantiques sont incorrectes : le sujet emprunte des opérateurs à l'espace problème de la machine à écrire et les applique à l'espace de l'édition de texte, ce qui entraîne des résultats inattendus et représente la cause la plus fréquente des erreurs. Finalement, cette analogie supposée parfaite par l'opérateur entre les deux situations va l'empêcher d'explorer d'autres manières de trouver des solutions.

4) *De l'incidence de l'enracinement des apprentissages sur le raisonnement analogique.*

Si une connaissance représente un moyen pour en acquérir une autre, plus pertinente pour la tâche à réaliser, elle peut aussi constituer un blocage à cette acquisition. Un tel phénomène à été signalé par Leplat (1988) chez les travailleurs vieillissants qui ont de la peine à abandonner des représentations familières qu'ils savent très bien faire fonctionner, pour des représentations plus adaptées, mais qu'il faudrait élaborer. Ces opérateurs éprouvent ainsi une difficulté à déstructurer certaines connaissances, ancrées profondément dans leur mémoire, pour leur en substituer de nouvelles, plus appropriées. En ce sens, on pourrait parler de rigidification, et même de dégradation des connaissances. On trouve fréquemment ces comportements chez des personnes qui n'ont été formées que sur un seul type

de dispositif et qui doivent, subitement, apprendre un nouvel outil. Le parallèle peut être fait avec la situation des informaticiens qui programment depuis les années 80 sur les mêmes systèmes (site-central et programmation procédurale) et à qui on impose brusquement de nouveaux environnements de développement (client serveur et conception orientée objet). Ils réagiront très certainement de la même manière que les travailleurs observés par Leplat.

5) *De l'influence des biais cognitifs sur le raisonnement analogique*

Stacy et Mancmillian (1995) ont mis en évidence un certain nombre de biais cognitifs intervenant dans le processus de compréhension et de résolution informatique. Ils suggèrent que l'utilisation de ces biais –*de représentativité, de disponibilité et de confirmation*– est à l'origine de traitements erronés et de mauvaises représentations du problème. Ces biais inciteraient les informaticiens à réutiliser des connaissances et des solutions inappropriées ; en somme, à effectuer des transferts négatifs.

Le *biais de représentativité* conduit ainsi les informaticiens à généraliser des phénomènes qu'ils estiment fréquents, sur des cas spécifiques (*par exemple, les caractéristiques globales d'une fonction se retrouvent dans une séquence spécifique de programme*), ou inversement, à prétendre qu'une observation spécifique représente un phénomène général.

Stacy et Mancmillian (1995) ont observé des informaticiens chargés d'évaluer un programme qui présentait une caractéristique originale : le nom de certaines variables était assez long. Les examinateurs se focalisèrent d'emblée sur cette particularité et leur rapport surestima largement la présence de ce type de données. En fait, parce que cette spécificité sortait des schémas classiques de programmation, les examinateurs considéraient ces quelques cas de variables comme une marque caractéristique du code évalué.

Pour le *biais de disponibilité*, les personnes évaluent la probabilité d'un événement de programmation par la facilité avec laquelle les exemples et les solutions peuvent être rappelées dans leur mémoire de travail (MCT).

Durant une étape de conception, ce biais va par exemple conduire des personnes à surestimer les avantages de certaines solutions informatiques parce qu'elles sont

plus facilement remémorables, et inversement, à sous-estimer les avantages de certaines autres parce qu'elles sont plus difficilement accessibles à l'esprit.
Dans un étude effectuée en milieu naturel par Teasley, Leventhal et Rholman (1993) durant le test et le debboguage d'un programme, les auteurs remarquent que dans la majorité des situations observées, une simulation exhaustive de tous les états possibles du programme est impossible à effectuer. Aussi, les informaticiens vont-ils déterminer un échantillon de cas à tester par ce biais de disponibilité : les cas évoqués représentent les situations critiques le plus souvent rencontrées dans leur pratique, même si ceux-ci ne correspondent pas forcément à la vocation du programme. Cette étude met également en évidence un autre biais : celui de confirmation.

Le *biais de confirmation* entraîne l'informaticien à rechercher et à réutiliser tous les éléments, toutes les solutions qui permettent de confirmer ses attentes et présupposés, et à faire abstraction de tous ceux qui pourraient les infirmer.
Dans la même étude que précédemment, Teasley et al. (1993) montrent que parmi l'ensemble des cas choisis, les informaticiens sélectionnaient davantage les tests positifs que négatifs parce que leur but était justement de montrer que le programme fonctionnait, et non pas qu'il échouait. C'est ce que les auteurs appellent le biais du test positif. De plus, ils remarquent aussi que l'expertise de l'informaticien d'une part, et le contexte du problème, d'autre part, n'altèrent en rien le recours à ce biais de confirmation : les expert et les novices agissent de la même manière, et ni l'exhaustivité des spécifications, ni la présence de "buggs"[23] dans le programme ne modifient son usage.

En fin de compte, deux facteurs peuvent expliquer le recours à de tels biais : d'une part, l'expérience de l'informaticien qui fournit un capital de connaissances et de solutions réutilisables rapidement et à moindre coût, et, d'autre part, le maintien d'une charge mentale acceptable durant la conception. En effet, mettre en œuvre de nouvelles représentations, les faire fonctionner, en tirer de nouvelles procédures constituent un effort coûteux que l'on essayera parfois d'éviter en continuant à travailler sur des représentations plus simples, comportant un petit nombre d'états

[23] Erreurs dans la rédaction d'un programme entraînant des pannes de fonctionnement informatique

ou d'opérations bien connues, même si celles-ci ne correspondent pas forcément à la nature de la tâche.

5.3.4 Les transferts intra-domaine et les transferts inter-domaine

On parlera de transferts intra-domaine pour tous les cas où les situations sources et cibles appartiennent au même domaine. Ces transferts sont parfois présentés comme ayant des effets positifs ou négatifs. Ainsi Détienne (1991b) a observé une réutilisation intra-domaine dans un problème de conception orientée objet : la solution de conception développée sur le projet courant est utilisée plus tard comme une source pour le développement d'autres solutions cibles. De même, Anderson et Thomson (1989) ont montré que pour écrire un programme en LISP, l'informaticien se sert d'un exemple d'utilisation d'une autre fonction qu'il a vue auparavant et qui a un effet analogue.

Autre transfert possible, celui inter-domaine où le sujet se réfère à une solution provenant d'un domaine source qui est différent du domaine cible.
Sebillote (1993) a examiné la façon dont les employés de bureau utilisaient les schémas d'action correspondant à des tâches d'un certain domaine pour en traiter une nouvelle dans un autre domaine. La tâche connue consiste à louer une voiture pour une mission et la tâche nouvelle à préparer un contrat de vente immobilière. Les sujets devaient simuler l'activité d'une secrétaire travaillant dans une agence immobilière. Le processus de construction de nouveaux schémas susceptibles de répondre à la nouvelle tâche s'établissait de la façon suivante :

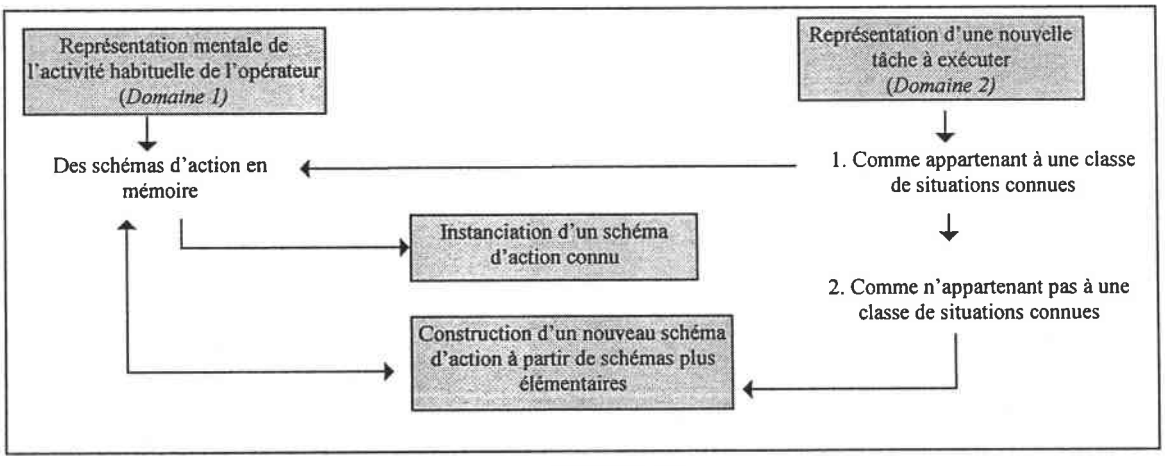


Figure 19
Schéma de résolution analogique inter-domaine

La tâche connue comprend trois schémas d'action séquentiellement ordonnés : "*Demander une voiture*", "*Faire la réception d'un billet*", "*Sortir un ordre de mission*".

L'examen de la tâche habituelle, décrite par les secrétaires, et celui de la tâche nouvelle révèlent que des sous-buts de la première sont utilisés pour exécuter la seconde. La nouvelle tâche comprend quatre schémas d'action, le premier étant "*préparer le contrat*". Ce schéma d'action, tel qu'il est prescrit dans la nouvelle tâche, nécessite de remplir un bordereau. Ce sous-but correspond dans la tâche connue à un sous-but dans le premier schéma d'action "*faire un bordereau*". Ce sous-but, commun aux deux tâches, va donc être utilisé comme premier schéma d'action au cours de l'exécution de la nouvelle tâche.

Au final, les transferts analogiques inter-domaine se révèlent avantageux, si les structures des situations problèmes (en termes de prérequis et de buts) sont équivalentes entre elles, et si les schémas d'action qui vont être dérivés correspondent globalement à la solution attendue. Sinon, le sujet s'exposera à des transferts négatifs.

5.3.5 En résumé

En définitive, il apparaît que la compréhension du problème, la résolution et l'apprentissage sont facilités si le sujet dispose déjà de connaissances (domaine source) qu'il peut réutiliser dans une situation analogue (domaine cible).

On a vu en particulier que l'élaboration d'une solution reposait sur la combinaison entre deux types de raisonnement.

a) Dans le premier, un schéma en mémoire est récupéré et, le concepteur le particularise pour en créer une nouvelle instance ("reasonning from a schema") (théorie de la récupération et approche pragmatique).
b) Dans le second, le nouveau problème/solution est mis en correspondance avec un problème/solution analogue pour générer une solution analogue ("reasoning from an analog"), sans, pour autant, qu'il y ait évocation préalable d'un schéma (raisonnement à partir de cas). (Détienne, 1991b).

Dans le cas des transferts inter-domaine, toutefois, le raisonnement analogique n'est efficace que si l'informaticien effectue des ajustements de structure et de sens. Sinon, la réutilisation peut freiner l'apprentissage et la résolution de problème.

En d'autres termes, dans toutes récupérations de solutions antérieures, le succès n'est jamais assuré. Des aspects non pertinents sont toujours présents dans la source, et l'informaticien doit savoir différencier ce qui est pertinent de ce qui ne l'est pas (Richard, 1990).

La partie qui va suivre a d'ailleurs pour objectif de déterminer les aspects des connaissances qui sont présentés par les études comme étant potentiellement transférables entre deux environnements de conception.

5.4 CARACTÉRISTIQUES DES COMPÉTENCES TRANSFÉRÉES

Jusqu'à présent, nous avons employé le terme générique de "*Connaissance*" pour désigner les connaissances de conception, les schémas mentaux, les solutions... réutilisés dans le cadre des raisonnements analogiques. L'utilisation de ce terme convenait tant que nous mettions l'accent sur les conditions de récupération de cette connaissance en mémoire.

Dans cette partie, notre objectif est d'identifier les savoirs susceptibles d'être transférés entre deux situations de conception informatique. C'est une perspective de recherche plus pragmatique que la précédente dans la mesure ou nous nous intéressons, cette fois-ci, aux connaissances professionnelle pouvant être réutilisées dans le cadre du travail. C'est pourquoi l'emploi du concept de "*Compétence*" de conception nous apparaît plus approprié car il revêt une signification plus opératoire (au sens d'adaptation à l'activité) et plus riche (par les domaines conceptuels qu'il recouvre) que le terme de connaissance. Nous allons revenir sur ces différents points en définissant tout d'abord ce qu'on entend par "compétence" en conception informatique.

5.4.1 Qu'est-ce que la compétence en conception informatique ?

On oppose deux définitions classiques de la compétence.

1) Traditionnellement en psychologie, le terme de compétence a été associé à celui d'aptitude ou de capacité, même si la distinction entre les deux reste ambiguë. La plupart du temps, l'aptitude est définie à partir d'une déterminisme génétique et on se trouve alors renvoyé à la polémique de la distinction entre les facteurs héréditaires et les facteurs de milieu dans les performances interindividuelles, même si aujourd'hui

on s'accorde sur l'idée d'une interaction entre les deux catégories de paramètres (Brangier et Tarquinio, 1997).

Cette première approche se relève peu appropriée pour définir la compétence dans le milieu de la conception informatique. Elle ne dit rien sur ses constituants et insiste peu sur le rôle formateur du contexte professionnel. Or, nous pensons justement que l'environnement technico-professionnel[24] agit sur la production de ces compétences, et affecte, du même coup, les conditions de leur transfert.

2) La perspective adoptée par l'ergonomie paraît plus en mesure de sérier ces aspects. La compétence a été introduite par Montmollin en 1984 comme «*des structures mentales où s'articule tout ce avec quoi l'opérateur réalise une tâche (ici considérée sous ses aspects cognitifs) : les connaissances sur le fonctionnement et l'utilisation des 'machines',... les représentations, mais aussi les savoir-faire, c'est-à-dire les types de fonctionnement (agglomérés parfois en routines), ainsi que les schémas stratégiques de planification des activités* ». En somme, la compétence recoupe des ensembles stabilisés de savoirs, de savoir-faire, de procédures et de types de raisonnement que l'on peut mettre en œuvre sans apprentissage nouveau.

D'un point de vue complémentaire, Leplat (1991) a distingué deux acceptions pour la compétence : béhavioriste et cognitiviste. La première définit la compétence par la tâche que le sujet sait exécuter ; son expression est liée au contexte. La seconde détermine la compétence par sa fonction heuristique : "*un système de connaissances qui permet d'engendrer l'activité*". Ce sont des métaconnaissances qui permettent de gérer et de juger les connaissances, et finalement, d'en générer de nouvelles.

Enfin, dans leur revue de questions, Barcenilla et Tijus (1998) rappellent que la compétence est également le produit de l'articulation entre des connaissances déclaratives (concepts et connaissances sur les faits du monde) et des connaissances procédurales (connaissances sur "*comment utiliser la connaissance*"). Ils remarquent aussi que les chercheurs restent divisés sur le type de relations qu'elles entretiennent ; soit une intrication sémantique, soit une distinction fonctionnelle entre elles :

- Dans le premier cas de figure, les deux auteurs évoquent Richard (1986) pour qui «*les connaissances procédurales concernent à la fois le savoir sur l'action*

[24] Nous faisons référence ici aux dispositifs techniques et au cadre socio-organisationnel qui fournissent les ressources à l'informaticien pour travailler.

qui peut être transmis par le langage et le savoir implicite, auquel nous n'avons pas accès, qui est directement lié à l'exécution des actions et qui se manifeste dans les habiletés sensori-motrices hautement automatisées » (p 34). De même, la théorie piagétienne de l'action articule à la fois des connaissances déclaratives et les connaissances procédurales : l'acquisition des connaissances résulte d'un « *échange continu d'informations entre la prise de conscience de l'action et la prise de connaissances de son objet* » (p 35)

- Dans le second cas de figure, Anderson (1983), postule deux modes indépendants de représentation : un mode de représentation pour les connaissances déclaratives, stockées sous forme de règles de propositions, et un autre mode pour les connaissances procédurales, stockées sous forme de règles de production. Pour George (1988), la distinction déclaratif/procédural se limite au mode de manifestation des connaissances : les connaissances déclaratives sont véhiculées par le langage, alors que les connaissances procédurales se manifestent ou sont acquises par l'action.

Sur la base de ces différentes approches, on pourrait dès lors dire que la compétence informatique remplirait les caractéristiques suivantes :

Elle ne peut se définir qu'en rapport direct aux tâches à accomplir et aux environnements technico-professionnels dans lesquels l'informaticien évolue : ces compétences sont apprises, consolidées et/ou ajustées de manière empirique sur le lieu même de travail. Mais elles peuvent aussi être acquises par l'intermédiaire de formations.

D'un point de vue fonctionnel, elles sont organisées en unités, c'est-à-dire coordonnées selon des hiérarchies ou des relations. On peut alors distinguer les connaissances qui permettent de comprendre "*comment ça fonctionne*" (connaissances déclaratives), des connaissances qui indiquent "*comment faire fonctionner ou concevoir*" (connaissances procédurales, savoir-faire, stratégies de résolution, schémas de planification, algorithmes) et les métaconnaissances qui génèrent de nouveaux savoirs sur la base d'anciennes connaissances (heuristiques). Cependant cette distinction est purement fonctionnelle, dans la mesure où ces connaissances sont associées par des rapports sémantiques plus étroits.

D'un point de vue opérationnel, ces compétences sont finalisées pour des tâches de conception précises (compétences en programmation, compétences pour la manipulation des dispositifs informatiques, compétences sociales pour la négociation...).

En somme, ces compétences informatiques regroupent des connaissances déclaratives, procédurales et métaconnaissances, qui sont elles-mêmes spécialisées pour des tâches spécifiques de conception (Cf.Figure 20).

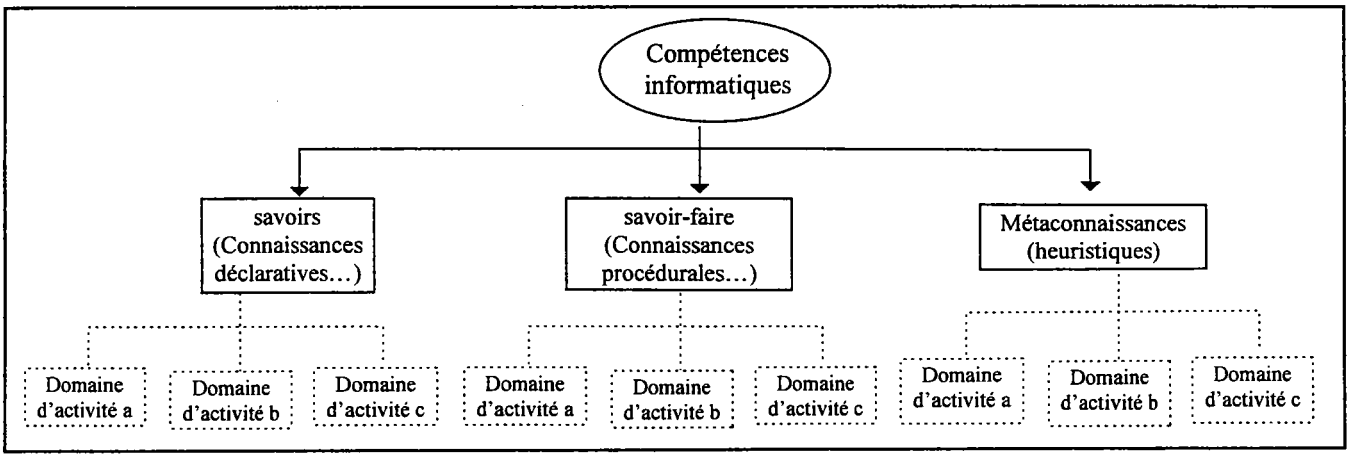


Figure 20
Les composantes des compétences informatiques

On pourra donc parler de transfert de compétences pour désigner les transferts de connaissances ou d'apprentissage entre deux environnements informatiques. Il reste à préciser maintenant quelles sont les compétences qui peuvent être réutilisées entre deux environnements informatiques.

5.4.2 Les différentes types de compétences transférables

Rappelons que le processus de raisonnement analogique se déclenche à chaque fois que l'informaticien s'initie à un second dispositif informatique, différent du premier. Les différentes recherches que nous avons recensées indiquent la transmissibilité possible de quatre types de compétences. Il s'agit :

1. des connaissances de programmation,
2. des stratégies de résolution,
3. des plans de conception,
4. de modèles mentaux.

a) Les connaissances de programmation transférées

Van Zuylen (1991) a identifié trois types de connaissances de programmation qui, avec plus ou moins de succès, pouvaient être transférés entre deux langages :

1. En premier lieu, les *connaissances syntaxiques* des langages (équivalentes aux connaissances déclaratives) représentent les mots, les expressions, le lexique... que le développeur doit maîtriser pour programmer correctement. Selon le niveau de similarité entre les langages sources et cibles, l'informaticien aura la possibilité ou non de réutiliser le vocabulaire. Par exemple, ce transfert sera envisageable entre C et C++ mais impossible entre C et smalltalk (lexique trop éloigné).
2. En deuxième lieu, les *connaissances sémantiques et pragmatiques* des langages (proches des connaissances procédurales) désignent les règles de d'écriture du langage sur lesquelles reposent le paradigme de conception (procédural, déclaratif, orienté objet), l'élaboration des algorithmes, les schèmes d'implémentation... La proximité paradigmatique de certains langages peut alors favoriser certains transferts : envisageables entre des langages procéduraux, mais impossibles entre un langage procédural et un langage déclaratif.
3. En dernier lieu, des *connaissances fonctionnelles* représentent les savoirs détenus par l'informaticien sur le domaine d'activité à informatiser. Il s'agit d'informations concernant le métier et les tâches de l'utilisateur final (par exemple, les métiers de la banque, de l'assurance...). Quel que soit les langages employés, le domaine reste le même.

Pour Van Zuylen, seul le dernier type de connaissances (fonctionnelles) est indépendant du langage, et serait à ce titre plus facilement transférable d'un langage vers un autre. Cependant, l'auteur remarque aussi l'existence d'algorithmes d'implémentation qui ne dépendent d'aucun langage de programmation en particulier. Confronté à un nouveau langage, le développeur aurait alors la possibilité de les reconduire par des transferts d'apprentissage positifs.

Harvey et Anderson (1996) ont étudié le rôle des connaissances déclaratives dans les transferts d'apprentissage de Lisp vers Prolog. Pour ces auteurs, ces connaissances ont

une fonction essentielle en fournissant des canevas généraux pour encapsuler et intégrer de nouvelles connaissances plus détaillées. De plus, ces connaissances déclaratives fournissent une analogie convenable pour guider l'acquisition d'informations procédurales. Leur étude suggère enfin que l'assimilation de nouvelles connaissances est facilitée lorsqu'une structure d'intégration a déjà été acquise grâce aux exemples trouvés dans le programme d'origine ou dans la documentation associée.

Outre les transferts de connaissances, la transmission des plans de résolution entre deux environnements de conception a également été mise en évidence.

b) Les plans de résolution

Dans des études inter-langage, Scholtz et Wiedenbeck, (1990, 1993) ont montré que certains plans de résolution élaborés pour un langage source sont transférables, et qu'ils peuvent même servir de guide à l'élaboration de solutions dans le langage cible appris. En particulier, deux modes de planification sont susceptibles d'être réutilisés :

- Un plan *tactique* : ce plan définit les stratégies locales pour la résolution de problèmes particuliers qui surgissent lors du développement. Ces problèmes sont généralement exprimés dans des concepts qui sont propres aux informaticiens (représentations internes au sujet ou "*Input*") ; ce qui les rend relativement indépendants des langages et donc potentiellement transférables.
- Un plan *d'implémentation* : ce plan a pour fonction de convertir la solution conceptuelle et le plan tactique de l'informaticien en solution technique. Il est donc dépendant du langage du dispositif d'accueil.

En somme, le nouveau paradigme oblige l'informaticien à revoir ses stratégies d'implémentation et à effectuer une boucle itérative d'ajustement entre plan tactique et plan d'implémentation.

Pour Samurçay (1987), il existe trois sortes de plan employées au cours de l'activité de conception, et qui sont plus ou moins liées au langage de programmation utilisé :

- *les plans stratégiques*, indépendants du langage, permettent d'identifier à partir de l'analyse du problème, le niveau global de l'algorithme à programmer ;
- *les plans tactiques*, indépendants du langage, permettent de raffiner les plans définis au plus haut niveau et de générer une solution adaptée au contexte de conception ;
- enfin, *les plans de mise en oeuvre,* proches des plans d'implémentation précédemment identifiés par Scholtz et Wiedenbeck (1990, 1993), sont dépendants du langage. Ils permettent de spécifier les unités de programmation pour un langage donné.

De sorte que lors d'un changement de langage, l'informaticien devra s'efforcer d'ajuster le plan de mise en oeuvre aux caractéristiques du nouveau paradigme ; les plans stratégiques et tactiques conservant, pour leur part, leur état d'origine.

Enfin, une dernière série d'études a montré que la planification de l'activité divergeait selon la nature du paradigme informatique utilisé (Détienne 1990a, Brangier et Bobiller Chaumon, 1995, Bobillier Chaumon, 1995) . Si une organisation planifiée de la tâche était plutôt déployée dans les environnements procéduraux, la démarche opportuniste était surtout mise en œuvre dans la programmation objet : en effet, la réutilisation d'objets informatiques impliquait souvent la remise en cause du plan de développement prévu initialement. Aussi, changer de paradigme de programmation – du procédural vers l'orienté objet– implique également de modifier la gestion et l'organisation de son activité.

Un certain nombre de stratégies de résolution ont également été décrites comme étant tantôt spécifiques, tantôt communes à différents paradigmes informatiques.

c) Le transfert des stratégies de résolution

Katz (1991) a étudié le transfert de stratégies de résolution entre Pascal (langage procédural) et LISP (langage déclaratif). Les sujets devaient écrire un même programme avec ces deux paradigmes différents. L'auteur, après avoir analysé les

transferts en oeuvre, en a déduit que les sujets employaient deux méthodes très différentes dans la résolution du problème : ceux développant avec Pascal étaient tentés d'utiliser préférentiellement une stratégie de résolution par "structures séparées" ; c'est-à-dire qu'ils dissociaient les entités qui relevaient des données et des procédures avant de s'atteler à la programmation. Les sujets expérimentés en LISP étaient plutôt enclin à associer les données et les procédures relevant d'un même processus. En analysant le passage du paradigme procédural vers le paradigme déclaratif, Katz a observé que les sujets commettaient de nombreuses erreurs de programmation parce qu'ils reproduisaient, de manière illicite, des stratégies de résolution procédurales dans le nouveau paradigme déclaratif.
Cette étude montre donc bien que chaque paradigme réclame une stratégie de résolution particulière, qui ne saurait être la simple transposition de celle apprise dans un autre contexte de programmation.

Ces conclusions sont d'ailleurs confirmées par les travaux de Lee (1994) et de Pennington, Lee et Rehder (1995). Ils ont analysé les conséquences du transfert des stratégies de résolution développées pour un paradigme procédural vers un paradigme objet (OO). Les observations montrent que les développeurs novices en OO (mais expérimentés en procédural) concentrent leur temps de travail à analyser le problème afin d'identifier les données et les procédures de programmation. Alors que les experts en OO passent plus de temps à comprendre le problème pour spécifier et classer les données abstraites (objets et méthodes associées) correspondant au domaine du monde réel. En fait, les novices répliquent en OO la démarche qu'ils avaient l'habitude de réaliser en conception procédurale. De plus, Lee (1994) observe que les stratégies d'évaluation des solutions progressent entre ces deux paradigmes : de "*largeur d'abord*" avec le langage procédural, c'est-à-dire que l'informaticien examine rapidement et superficiellement toutes les solutions possibles, ces stratégies sont menées en "*profondeur d'abord*" avec le paradigme objet ; l'informaticien explore les problèmes en évaluant précisément une seule solution à la fois.
Sholtz (1991) a réalisé une expérience dans laquelle il demandait à des sujets, expérimentés en Pascal, de résoudre un problème en langages ADA et en Icon. Il a remarqué que les modifications effectuées sur les schémas de résolution récupérés étaient moins nombreuses entre Pascal et ADA qu'entre Pascal et Icon. Ces

changements touchaient les schémas de haut niveau (tels que les plans stratégiques de résolution), les schémas algorithmiques (tels que les plans tactiques) et les schémas de bas niveau (telles que les connaissances sémantiques et syntaxiques).

Au final, ces différentes études suggèrent fortement le rôle formateur des paradigmes de conception dans la configuration des stratégies de résolution développées.

Petre (1990) apporte toutefois un jugement plus nuancé. Elle affirme que "*si le langage de programmation peut entraver ou faciliter l'expression d'une solution, celui-ci, en revanche, n'influence pas directement sa nature*". En d'autres termes, le paradigme affecte le transfert des solutions et non la solution elle-même. L'expert tend a réutiliser l'ensemble de ses stratégies de résolution. Cette procédure ne serait remise en cause que s'il éprouve des difficultés lors de l'implémentation des solutions. Auquel cas, il aura le choix entre ajuster ses solutions ou en construire de nouvelles.

Finalement, tous ces résultats semblent indiquer que les stratégies de résolution sont potentiellement transférables lorsqu'elles sont compatibles avec le paradigme cible. Sinon, les informaticiens devront ajuster leurs stratégies, au risque de réaliser des transferts négatifs.

d) Les modèles mentaux transférables

- Qu'est ce qu'un modèle mental ?

Il faut d'abord distinguer le modèle mental qui a une structure figée et qui est stocké en mémoire à long terme, d'une représentation mentale qui est localisée en mémoire de travail (mémoire à court terme) et qui est élaborée au cours de l'activité pour supporter l'action individuelle (Iosif, 1993). Leplat (1988) parle aussi de représentations fonctionnelles. La structure de ces représentations est plus fragile et plus labile que celle du modèle mental car elles évoluent au gré des circonstances de la résolution et disparaissent une fois les objectifs atteints.

Un modèle mental correspond, selon la définition de Norman (1983) à «*ce que l'individu a réellement dans la tête et ce qui le guide dans l'usage qu'il fait des choses* ». Ainsi, en interagissant avec les différents dispositifs de conception informatiques (langages, outils de développement...), les informaticiens forment des représentations internes ou des modèles mentaux qui sont des sortes de mode d'emploi

de ces outils (Comment les utiliser ? Dans quel cadre ? Dans quels buts ? Sous quelles conditions ?...) Johnson-laird, 1989. Pour Rasmussen, 1986, le modèle mental représente les connaissances de l'opérateur sur le système technique qu'il contrôle : des connaissances sur les propriétés fonctionnelles, causales (la structure relationnelle) du système et sur le contenu du travail. Elles servent à la planification de l'activité et au contrôle des actes lorsque ces connaissances sont activées par l'observation de l'état actuel du système.

Hoc (1987a et b) propose un modèle mental spécifique à la conception informatique ; le Système de Représentation et de Traitement (SRT) : « *c'est un système particulier défini par un certain type de représentations, lui même relié à un autre type de traitements* ». Ce SRT informatique est constitué de schémas et de plans et fournissent aux sujets des structures assimilatrices nécessaires pour le développement de leur activité ultérieure. Il existe un SRT par type d'environnement informatique utilisé.

En résumé, les modèles mentaux servent à guider l'activité des sujets, à interpréter l'état du système et à faire des prédictions des états futurs du système.

- A quoi servent ces modèles mentaux dans la conception ?

Pour démontrer la fonction de ces modèles mentaux dans la conception informatique, Canas, Bajo et Gonzalvo, (1994) ont réalisé une expérience dans laquelle ils montraient à des informaticiens novices le déroulement d'un calcul en temps réel sur un ordinateur : "*données en entrée, traitements, appel d'informations complémentaires en mémoire, traitements, données en sorties*". Puis, les auteurs ont demandé à ces novices de réaliser un programme qu'ils ont comparé à celui développé juste avant l'expérience. Les résultats indiquent que la seconde version se révèle plus performante que la première. Pour Canas et al., le fait de rendre visible le fonctionnement de l'ordinateur enrichit le modèle mental de l'informaticien, et conduit ce dernier à développer un programme qui prend davantage en compte les contraintes fonctionnelles et techniques de la machine.

Dans son expérience, Visser (1988) a analysé l'activité psychologique sous-jacente d'un informaticien engagé dans la mise au point d'un programme de commande d'une installation automatisée. Pour ce faire, l'auteur a observé durant cinq semaines l'activité réelle d'un opérateur chargé de mettre en conformité le programme informatique aux spécifications de la production. L'analyse des données montre que la

stratégie de mise au point consiste essentiellement, pour l'informaticien, à élaborer un modèle mental de ce qu'il pense être le programme correct ("*la référence*" récupérée de ses expériences passées) pour le comparer à la représentation du programme qu'il a à corriger (représentation construite durant l'activité). C'est la confrontation de ces deux modèles qui permettra à l'informaticien de diagnostiquer les erreurs.

- *Ces modèles mentaux sont-il transférables ?*

Dans leur revue de questions sur les modèles mentaux, Straggers et Norcio (1993) soulignent la divergence des points de vue :

- Pour une première série de chercheurs (Norman, 1987 ; Gentner et Gentner, 1983), les individus ne construisent pas leurs modèles mentaux *ex nihilo,* mais récupèrent et dérivent ceux qui leur semblent les plus appropriés à la situation problème. Autrement dit, le transfert des modèles mentaux serait tout à fait possible.
- Pour d'autres auteurs, au contraire (Moran, 1981), un modèle mental ne peut être développé ou inféré sans une phase d'apprentissage préalable. Dans ce cadre, c'est l'acquisition de nouvelles informations et de nouvelles données qui permet de construire un modèle mental, et non le transfert d'anciennes structures de connaissances.

D'un point de vue fonctionnel, le transfert de ces modèles mentaux simplifie l'activité de raisonnement sur le principe de "*l'économie cognitive*" : on ne reconstruit pas ce qui existe déjà. Toutefois, ces transferts peuvent aussi générer du stress et une charge de travail supplémentaire lorsque le modèle récupéré n'est pas adapté à la situation. L'informaticien encoure alors le risque d'interpréter les états du système à travers des représentations obsolètes ou inadéquates. Straggers et Norcio (1993) désignent ainsi par "*Lock-up cognitif*" (blocage cognitif") la conduite qui consiste à ne pas remettre à jour son modèle mental dans un nouveau contexte de travail. Par exemple, des informations nouvelles et originales peuvent être négligées parce que le modèle mental transféré ne permet pas de les identifier. De même, le sujet peut développer des conduites d'interaction homme-machine inappropriées, car dictées par un modèle mental anachronique. L'expérience de Douglas (1983) (déjà citée) sur les analogies présupposées entre "machine à écrire" et "traitement de texte" montre d'ailleurs les effets de tels transferts.

Toutefois, comme l'ont aussi indiqué Lange et Moher (1989), les développeurs expérimentés ont à leur disposition des modèles mentaux plus matures et plus riches qui leur donnent la possibilité de pallier les éventuelles incomplétudes ou inexactitudes de ces transferts négatifs.

5.4.3 En résumé

Des compétences peuvent donc être transférées d'un dispositif de conception vers un autre. Parmi celles-ci, se distingue :

- *Un niveau conceptuel* : selon le langage cible, certaines structures de connaissances (procédurales et fonctionnelles) sont réutilisées avec plus ou moins de facilités et de succès entre deux paradigmes de programmation.
- *un niveau organisationnel* : les plans tactiques qui structurent l'activité mentale de résolution peuvent être employés concurremment dans deux contextes de programmation différents. Ils ne se rattachent donc à aucun langage. En revanche, les plans d'implémentation qui organisent la conversion de la solution dépendent directement du paradigme informatique. A ce titre, ils doivent subir des ajustements lors d'un changement d'environnement.
- *un niveau de résolution* : Selon les auteurs, tout ou partie des stratégies de résolution peuvent être réutilisées dans un nouvel environnement de conception. Toutefois, les processus de représentation du problème et de définition des données, ainsi que les traitements nécessaires à la conception seraient spécifiques à certains langages (opposition entre langage procédural et OO).
- *un niveau de représentation* : les modèles mentaux peuvent servir concurremment dans deux contextes de programmation différents, si toutefois, ces derniers sont techniquement et fonctionnellement proches. Sinon, le transfert risque de générer des erreurs de conception et une charge mentale plus élevée.

Finalement, ces transferts cognitifs n'induisent pas les mêmes effets selon les langages de programmation employés : plus les paradigmes sont éloignés les uns des autres, plus le risque de transferts inappropriés (ou négatifs) est important. Les difficultés d'apprentissage et de compréhension du problème, ou encore les erreurs de résolution et de programmation, pourront être l'expression symptomatique de ces

"dysfonctionnements" cognitifs. A l'inverse, les transferts positifs favoriseront l'apprentissage du nouveau paradigme et rendront plus efficiente la conception. Enfin, on rappellera que le contexte de la conception est un artefact du transfert d'apprentissage : plus les problèmes se ressemblent d'une situation à l'autre, plus grande sera la probabilité que le développeur instancie les pratiques cognitives associées au langage source.

6. CONCLUSION

Une première série d'études réalisées sur l'activité de programmation nous a permis d'appréhender les formes prises par la conception informatique en tant qu'elle consiste à conduire un projet d'une analyse des besoins à un programme informatique. Sous cet angle, l'activité de conception est conçue comme une organisation hiérarchisée et/ou opportuniste d'étapes de conception, dont le déroulement reste guidée par les circonstances et les découvertes de l'activité, ainsi que par le principe d'économie cognitive.

D'autres études ont plus particulièrement mis l'accent sur le fond, en s'intéressant aux stratégies mises en oeuvre par les informaticiens dans l'activité de programmation pour produire une application. Sous cet angle, l'activité de programmation est vue comme une tâche qui s'articule principalement autour de la résolution de problèmes et de la transformation d'une solution conceptuelle en une représentation technique acceptable et implémentable.

Enfin, une dernière série de recherches s'est principalement intéressée aux processus cognitifs intervenant lors de l'apprentissage d'un second langage informatique. Sous cet angle, l'activité de programmation est perçue comme l'actualisation de connaissances et de stratégies préexistantes aux circonstances et aux exigences du nouveau paradigme de développement.

Bien que ces différents travaux aient contribué dans l'ensemble à améliorer les connaissances sur le métier de concepteur informatique, il existe néanmoins deux réserves qui nous autorisent à relativiser la portée et la généralisation de ces résultats. L'une concerne la méthodologie, l'autre touche aux aspects épistémologiques.

Au niveau méthodologique, la plupart de ces travaux ont été effectués dans un cadre expérimental où toutes les variables sont rigoureusement contrôlées. L'activité réelle de programmation n'est donc malheureusement reproduite que de manière partielle et/ou incomplète. Ainsi, les problèmes posés aux informaticiens sont-ils "*bien définis*", alors que dans la réalité, les projets informatiques sont souvent flous et mal définis. Ces problèmes apparaissent également assez peu représentatifs de ceux habituellement traités par les informaticiens (gestion d'une compétition de piscine dans un cas, organisation informatique du stock d'un magasin dans un autre cas...).
Les chercheurs font également volontairement abstraction des aspects "*contingents*" du travail. Ainsi la division et la répartition du travail, la communication dans et entre les projets, les interaction entre les individus, les modèles de relation au travail, la culture professionnelle, la pression des clients... représentent autant de variables qui orientent l'activité informatique et structurent les modes de résolution.
Enfin, la population étudiée correspond presque toujours à des débutants en informatique (voir à des étudiants) car il est plus aisé d'identifier les mécanismes de raisonnement mis en jeu par ce type de sujets. Or, ces travaux précisent rarement si les conclusions dégagées pour les novices restent valables pour une population d'experts en programmation.

Pour ces raisons, nous estimons que la démarche expérimentale ne permet pas, "*toutes situations étant égales par ailleurs*", d'appréhender globalement les mécanismes cognitifs mis en oeuvre dans l'activité effective de conception. Par "activité effective", nous entendons celle qui se déroule réellement et quotidiennement dans le service informatique d'une entreprise. L'expérimentation de laboratoire, bien qu'indispensable à la production scientifique, livre plutôt une représentation artificielle de la tâche de programmation, dont les résultats et les observations n'ont de validité que dans le cadre restreint de ces variables contrôlées.

Du coup, une remarque épistémologique s'impose : quel est le statut des cognitions décrites en tant qu'elles relèvent uniquement de l'utilisation des outils techniques et non du contexte de leur production et de leur utilisation ? Au-delà des raisons techniques, quels sont les facteurs qui conditionnent réellement la conception informatique ? L'ensemble des travaux présentés ci-dessus ne peut répondre à ces

questions car les chercheurs s'en sont généralement tenus à décliner ces processus cognitifs sans tenir compte des facteurs sociaux et organisationnels affectant l'activité de l'informaticien. Ces dimensions du travail, qui ne sont pas strictement des stratégies cognitives de résolution de problème, les conditionnent pourtant de manière importante.

C'est pourquoi le caractère appliquée de notre recherche devrait apporter une validation écologique à ces processus. Notre approche se propose en effet d'examiner les raisonnements de conception selon leur contexte de production. L'environnement technique est perçu comme un artefact qui structure les savoirs des opérateurs. C'est donc dans une approche située de la cognition que nous nous plaçons : la signification des conduites humaines ne pourra être établie qu'en référence à leur insertion dans le cadre des environnements technologiques utilisés.

C'est dans cette perspective que se situe notre travail de recherche sur le terrain.

CHAPITRE III

Le travail de recherche sur le terrain

RESUMÉ

Notre travail de terrain s'inscrit dans le cadre des recherches menées sur l'analyse des stratégies individuelles de conception et sur l'étude des transferts de compétences. Nous montrons que ces compétences peuvent s'évaluer en dressant le modèle mental qu'a l'utilisateur du dispositif. Dans cette optique, nous présentons un modèle d'analyse psycho-cognitif nous permettant d'appréhender l'activité mentale de l'informaticien durant sa tâche de maquettage, et ainsi, d'aboutir à son modèle mental.

L'organisation de ce chapitre comporte trois parties

1) Présentation du cadre méthodologique
Après avoir exposé notre problématique et nos hypothèses, nous présentons les concepts théoriques à partir desquels nous construisons notre modèle d'analyse. Selon la démarche méthodologique établie, il va nous permettre de dresser et de comparer les modèles moyens détenus par trois les types d'échantillon (Experts Pacbase - Expert NSDK et Novices NSDK). Nous décrivons aussi les caractéristiques du terrain où s'est déroulée l'enquête. Nous précisons enfin la méthodologie déployée.

2) Présentation des résultats:
Cette partie débute par la description du système homme-machine relatif à chaque environnement de conception. Nous sommes ainsi amenés à comparer :
- *a) les caractéristiques fonctionnelles et techniques des environnements source (Pacbase sur site-central) et cible (NSDK en client serveur) ;*
- *b) l'organisation de l'activité de maquettage observée dans chaque contexte de développement ;*
- *c) l'analyse des contraintes et des astreintes générées par les nouvelles caractéristiques techniques.*

3) La dernière partie de ce chapitre est consacrée à la définition des modèles mentaux de conception des informaticiens, par l'application du modèle d'analyse. A partir des données recueillies sur les trois groupes de développeurs, nous examinons la nature :
- *a) des démarches de structuration de l'activité (modèle d'action),*
- *b) des stratégies de résolution mises en œuvre (modèle de résolution)*
- *c) de la qualité de la relation homme-machine (modèle d'interaction),*
- *d) des connaissances et des exemples de référence employés pour concevoir les interfaces (modèle de référence).*

1. CADRE MÉTHODOLOGIQUE

Notre démarche porte sur l'analyse des stratégies individuelles de conception, et plus particulièrement sur l'étude des transferts d'apprentissage dans le cadre de changements technologiques. Outre l'importance de la détermination de ces raisonnements, l'analyse des conditions du transfert de ces compétences est un élément essentiel de la compréhension des difficultés rencontrées par les débutants. Quant à notre méthodologie, elle concerne la comparaison des raisonnements détenus par trois échantillons grâce à un modèle d'analyse psycho-cognitif, et s'appuie sur la présentation du terrain et de l'objet d'étude.

1.1 PROBLÉMATIQUE ET HYPOTHÈSES

1.1.1 Problème posé

Le problème auquel nous nous intéressons est celui du transfert des compétences entre deux environnements de conception informatique. Plus exactement, il s'agit de déterminer dans quelle mesure les transferts de compétences d'un environnement de conception site-central source vers un environnement client serveur cible peuvent être tenus pour responsable des difficultés que rencontrent des informaticiens novices dans leur nouvelle tâche de conception.

Dans ce but, les approches, études et modèles présentés dans le cadre théorique de notre recherche ont permis d'établir que :

1. L'activité de conception impliquait fondamentalement un processus cognitif de résolution de problème se déroulant suivant plusieurs étapes clefs ; compréhension du problème, mise en œuvre de connaissances, définition de stratégies de résolution et réutilisation d'anciens schémas de connaissances, régulation opportuniste ou hiérarchique de l'activité de conception, implémentation de la solution finale. Mais, les études ont aussi montré que, selon les paradigmes de conception considérés, la nature ainsi que les conditions de mise en œuvre de ces processus cognitifs pouvaient diverger.

2. Pour concevoir et interagir avec leurs outils techniques, les informaticiens disposent d'un ensemble de compétences informatiques, c'est-à-dire d'un système constitué de connaissances déclaratives, procédurales et de métaconnaissances qui, d'une part, sont finalisées pour des tâches informatiques précises et, d'autre part, modélisent les propriétés du dispositif technique. Ces systèmes de compétences correspondent à des modèles mentaux. Ils sont stockés en mémoire à long terme et sont acquises par l'expérience et les formations. Selon la proximité des paradigmes de conception source et cible, le transfert de certaines classes de compétences est plus ou moins indiqué. Dans le cas favorable, le transfert positif favorisera l'apprentissage et la résolution ; dans le cas défavorable, le transfert négatif nuira à la collaboration homme-machine et à l'activité de conception.

3. Le raisonnement analogique est un processus clef de la conception informatique, intervenant (i) dans la résolution –i.e. par la réutilisation de connaissances– et (ii) dans le réapprentissage d'un nouveau dispositif –i.e. en favorisant le transfert de connaissances–. La description de son fonctionnement nous a conduit à identifier un certain nombre de caractéristiques susceptibles de générer des transferts positifs ou négatifs entre deux domaines de conception source et cible (expérience de l'informaticien, rigidité des schémas de connaissances, biais de représentation, inadéquation des connaissances transférées…)

4. La mutation des technologies de conception conduit à un repositionnement des pratiques professionnelles de l'informaticien, notamment dans ses façons de penser, d'organiser et de réaliser son activité. De la sorte, un nouveau système technique n'est pas seulement une entité intermédiaire entre le sujet et l'objet. C'est un artefact qui transforme à la fois les formes et l'organisation du travail, ainsi que les contenus des savoirs et des savoir-faire des informaticiens.

5. Les dispositifs techniques[25] de conception sécrètent des manières de penser et d'organiser l'activité de conception, d'agir et d'interagir avec l'environnement de conception. Le passage d'une technique de conception vers une autre marque également l'évolution du contenu même de la tâche de conception : il s'agit moins de développer des programmes que de concevoir des situations d'interaction appropriées.

6. La conception n'est pas un processus isolé et décontextualisé ; elle est profondément située. Elle dépend et repose sur de multiples dimensions (technique, cognitive, collective, méthodologique…) qui forment l'environnement de conception , et qui affectent de manière plus fondamentale encore les conditions de sa réalisation et de son développement.

Mais, derrière ces six points se découvrent une série de questions plus fondamentales, en rapport avec le cadre de notre recherche. En particulier :

[25] et notamment, leur architecture site-central *Vs* client serveur ; leur logique événementielle *Vs* logique séquentielle ; leur interface textuelle *Vs* Interface graphique ; leur paradigme procédural *Vs* paradigme orienté objet

- Quels sont les rapports que le système cognitif entretient et développe avec les environnements de conception site-central et client serveur ?
- Dans quelle mesure les processus cognitifs doivent-ils être réorganisés au contact des nouveaux dispositifs client serveur ?
- Parmi l'ensemble des compétences développées par l'informaticien dans l'environnement site-central, comment distinguer les compétences qui sont appropriées de celles qui sont inadaptées au nouvel environnement client serveur ?
- Comment favoriser les transferts d'apprentissage positifs et endiguer les transferts négatifs entre les deux environnements ?
- Enfin, quelles origines doit-on donner aux difficultés de conception que rencontre l'informaticien dans l'environnement cible ? : est-ce le manque d'ergonomie des nouveaux dispositifs ou bien, s'agit-il davantage de transferts d'apprentissage inappropriés ?

De notre point de vue, la réponse à ces questions implique le recours à des théories[26] cognitives portant sur les modèles mentaux, les théories des schémas, le raisonnement analogique, les compétences et le modèle de la compatibilité cognitive. La partie qui suit présente ces concepts généraux .

Concepts généraux

En interagissant avec leur environnement de conception, les informaticiens ne sont pas passifs. Ils ont des objectifs, des intentions, et font des inférences et des anticipations à partir du fonctionnement de leur dispositif technique. Ils se construisent ainsi **un modèle mental** de ce dispositif. Ce modèle mental recoupe également plusieurs sous-modèles mentaux finalisés pour des tâches particulières (nous y reviendrons par la suite). De la sorte, l'informaticien n'interagit pas "directement" avec le dispositif mais avec le modèle mental qu'il s'est construit de ce dispositif.

Dans ces conditions, le changement d'environnement technique doit théoriquement conduire à un basculement de ces modèles cognitifs, c'est-à-dire à une reconfiguration

[26] Ces différentes approches théoriques ont déjà été largement présentées et débattues dans les deux chapitres précédents.

des schémas acquis en conception site-central selon les circonstances et les exigences du nouveau contexte de conception client serveur.

Sur certains aspects, cette transition "cognitive" peut se révéler facile et bien maîtrisée car les caractéristiques de l'environnement permettent de reproduire en l'état, ou avec quelques légères adaptations, les modèles mentaux développés pour l'environnement source. Sur d'autres aspects, au contraire, cette migration dévoilera des obstacles nombreux et variés qui mettront en évidence l'échec patent du transfert analogique : la rigidité de l'expérience, la persistance de certaines structures de connaissances obsolètes, les automatismes de conception, la mauvaise représentation de la situation, le déficit de compétences adéquates... seront autant de facteurs susceptibles de créer ce type de dysfonctionnement.

C'est pourquoi les erreurs d'utilisation de ces dispositifs, les difficultés de conception des programmes, les mauvaises interprétations des besoins utilisateurs... ne doivent pas être comprises comme étant dues au hasard, à des limitations de la capacité de la mémoire de travail ou à une inattention. Elles doivent être vues au contraire comme une inadéquation fonctionnelle entre les caractéristiques cognitives de l'informaticien (plus précisément de son modèle mental) et les spécificités de la nouvelle situation de conception. En d'autres termes, les problèmes de conception dans le nouvel environnement technique s'expliquent principalement par une incompatibilité des compétences évoquées par l'utilisateur (*via* le raisonnement analogique) avec le modèle de conception requis pour développer dans le nouveau dispositif.

Pour ces raisons, il nous semble que le point de passage obligé pour favoriser des transferts positifs réside dans la prise en compte des modèles mentaux de l'informaticien. En effet, leurs caractéristiques intrinsèques et la manière dont ils sont réactivés vont conditionner les modalités de formation au nouveau dispositif.

Dans cette perspective, nous développerons la notion de sous-modèles mentaux qui représentent les principaux schémas de compétences que possède l'informaticien pour concevoir dans un paradigme informatique particulier, et interagir avec un système informatique. Cette distinction est purement théorique car tous ces modèles interagissent entre eux, s'enrichissent mutuellement et composent ainsi le modèle mental relatif à une situation de travail donnée.

Changer d'environnement de conception revient à transformer ces sous-modèles dans des proportions plus ou moins grandes qui dépendent de la proximité physique et conceptuelle des environnements source et cible : plus ces écarts sont importants, plus les transformations sur ces modèles mentaux seront nécessaires. Il conviendra alors d'apprendre, et non plus de réapprendre, de nouvelles structures de compétences. Plus cette distance est réduite, plus les transferts d'apprentissage seront aisés et la formation rapide.

La thèse qui en découle consiste dès lors à supposer qu'en centrant les transferts d'apprentissage sur une réutilisation sélective de certaines structures de connaissances issues de l'environnement source (les plus aptes selon les caractéristiques du domaine cible), l'élaboration d'un modèle mental correct suivra. Autrement dit, le transfert d'habiletés acquises dans un situation donnée peut se formaliser en savoir-faire opérationnel, traduit dans de nouvelles compétences.
Faut-il encore repérer parmi les compétences du domaine source, candidates à un éventuel transfert vers le domaine cible, :

i) celles qui nécessitent un réajustement avant leur réutilisation ;

ii) celles pour lesquelles l'adaptation est inutile car transférables en l'état ;

iii) celles, enfin, dont le transfert doit être absolument proscrit car totalement inappropriées au domaine cible.

En définitive, il apparaît que les compétences transférables peuvent s'évaluer en dressant le modèle mental des informaticiens. Or, établir ce modèle mental revient à modéliser l'utilisateur qui le porte. C'est précisément l'ambition du modèle d'analyse psycho-cognitif que nous allons exposer. Son objectif est de rendre compte des conduites de conception que les informaticiens mettent en œuvre dans chaque environnement de développement. Mais il se présente également comme une grille de lecture des difficultés que rencontrent les concepteurs débutants en client serveur.

1.1.2 Présentation du modèle d'analyse psycho-cognitif

a) Principes de base

Plus haut, nous avons dit que les conduites de travail étaient gouvernées par des modèles mentaux qui définissent la charpente des stratégies choisies. Le concepteur est un être de desseins. Il créé, enregistre dans sa mémoire et en extrait des plans, des modèles, qui lui indiquent comment opérer s'il veut parvenir à ses fins et agir en accord avec les contraintes environnementales. Identifier ces modèles, ces théories d'action, c'est se donner les moyens d'accéder à une véritable compréhension des compétences et des raisonnements en œuvre dans de conception, et dans les transferts d'apprentissage.

La méthode utilisée pour déterminer ces représentations consiste à dresser un modèle moyen des informaticiens. Mais, pour cela, il faut le paramétrer avec des informations qui sont propres à chaque informaticien : ses connaissances, ses raisonnements, ses stratégies de résolution, ses modalités d'interaction avec le système, etc. C'est dans ce but que le modèle d'analyse psycho-cognitif a été conçu.

b) Le modèle d'analyse psycho-cognitif

A partir de différents concepts, théories et modèles[27] sur le fonctionnement cognitif de l'informaticien, nous avons déterminé quatre niveaux d'analyse permettant d'identifier les processus qui gèrent les principales conduites du concepteur. Il s'agit respectivement :

1. du sous-modèle d'action
2. du sous-modèle de résolution
3. du sous-modèle d'interaction
4. du sous-modèle de référence

[27] Les différentes théories dont nous nous sommes inspirés pour élaborer ces modèles sont exposées ici de manière très succincte. Le lecteur pourra les retrouver dans les parties correspondantes du cadre théorique.

1) le sous-modèle d'action

Ce modèle mental permet de déterminer si l'activité s'organise selon un plan prédéfini (aspect hiérarchique et planifié de l'activité), en déterminant dans le temps un certain nombre d'actions à réaliser et d'objectifs à atteindre (Hoc, 1987 ; Sacerdoti, 1974). Mais ce plan peut être aussi remis en cause dès que de nouvelles opportunités, techniquement ou "cognitivement" meilleures, se présenteront (aspect opportuniste de l'activité) (Hayes-Roth et Hayes-Roth, 1979 ; Suchman, 1987 ; Visser, 1988, 1989 et 1994). Dans le cadre de l'activité informatique, ce sous-modèle rend compte du niveau d'organisation et de régulation de l'activité de conception, selon les contraintes de chaque environnement.

2) le sous-modèle de résolution

Ce sous-modèle suppose différents processus cognitifs pour caractériser le problème et trouver sa solution. Il peut s'agir de :

- la *représentation* du problème et son diagnostic (Hoc et Amalberti, 1995) ;
- la *construction* d'une solution par *récupérations de schémas de connaissances* issus de sources internes (mémoire) ou externes (supports techniques) (Détienne, 1990) ;
- une construction *ad hoc* de la solution par des stratégies *d'essais/erreurs* ou de type *fins-moyens* (Hoc, 1987) ;
- la *comparaison et la sélection* des solutions en *largeur d'abord* ou en *profondeur d'abord* (Jeffries, Turner, Polson et Atwood, 1981) ;
- *L'évaluation des solutions envisagées* selon des critères techniques (performance du programme, temps de réponse...) mais aussi par rapport aux objectifs initiaux du projet (cahier des charges, besoin du client...) (Bonnardel, 1991).

Rapporté à la conception informatique, ce sous-modèle mental permet d'appréhender la représentation que l'informaticien se fait de son problème et des moyens qu'il met en œuvre pour sa résolution en tenant compte, bien évidemment, des paradigmes de programmation relatifs à chaque environnement de développement (procédural et séquentiel pour le site-central ; événementiel et orienté objet en client serveur).

3) Le sous-modèle d'interaction

Ce modèle fait référence aux systèmes de connaissances que possède l'informaticien pour interagir avec son environnement. Si ses connaissances sont suffisamment proches de celles requises pour utiliser le dispositif technique, alors l'informaticien se trouve en situation de compatibilité cognitive avec ce système. L'informaticien sera d'autant plus motivé à s'investir ou à s'engager dans la conception qu'il n'a pas à se préoccuper de la façon dont il doit utiliser le dispositif, et que le dialogue homme-machine est intuitif, harmonieux et naturel (Senach, 1990). En revanche, si l'informaticien doit se livrer à de multiples traitements afin d'interpréter les états du système ou réaliser des actions sur ce système (pour le codage d'instructions par exemple), alors les efforts déployés pour de tels traitements généreront une charge de travail supplémentaire qui entravera le processus d'engagement.

En somme, ce modèle mental permet d'une part, d'apprécier la qualité de la coopération homme-machine dans chaque environnement de conception utilisé et, d'autre part, de dégager des niveaux d'investissement dans l'acte de conception : l'informaticien sera d'autant plus motivé à s'impliquer que le système qu'il utilise est compatible avec ses modes de pensée.

4) le sous-modèle de référence

Ce dernier modèle envisage l'ensemble des représentations prototypiques à partir desquelles les composants informatiques (interfaces graphiques, programmes, objets...) vont être déclinés (Pair, 1988). Ce sont des schémas de connaissances correspondants aux éléments informatiques les plus caractéristiques de chaque contexte de conception. En fait, ces modèles de référence, sorte de «protoconcepts» (en référence aux travaux de Fodor, 1986[28]), recouvrent l'ensemble des propriétés et des fonctions d'une entité type de développement. Ils peuvent être inspirés directement de l'environnement technique, provenir d'anciens projets de développement, ou bien, avoir été repris de modèles extérieurs, comme, par exemple, les applications bureautiques.

L'informaticien s'appuie sur ces modèles conceptuels de programmation très généraux pour dégager des composants informatiques plus spécifiques. Ces

[28] Fodor (1986) "*La modularité de l'esprit*". Paris, Minuit.

composants sont une épure de ces modèles mais ils peuvent aussi être construits à partir de plusieurs fragments de différents modèles.

1.1.3 Hypothèses

Ces quatre sous-modèles apparaissent comme des bases conceptuelles qui peuvent aider à la compréhension de l'activité de conception et des phénomènes de transfert d'apprentissage. Ils fournissent un modèle de la cognition en tant que processus globaux de la conception : c'est-à-dire qu'ils constituent les structures de données et de traitements de base qui *i)* organisent la conception, *ii)* supportent la résolution, *iii)* guident l'interaction et *iv)* soutiennent l'apprentissage. Ces modèles mentaux sont élaborés pour un environnement spécifique, et sont donc particuliers à un contexte de conception déterminé.

De ce point de vue, le passage d'un environnement à un autre ne requiert pas seulement l'assimilation de nouvelles connaissances, mais l'ajustement des modèles mentaux antérieurs. Dès lors, les difficultés d'apprentissage et les erreurs de conception ne résideraient pas tant dans un déficit conceptuel que dans le transfert de modèles mentaux inappropriés, et aussi, dans la confrontation à des modèles de raisonnement et des formes d'acquisition auxquels les informaticiens ne sont pas familiarisés.

Ces aspects généraux nous conduisent à dégager quatre hypothèses spécifiques à l'utilisation et au transfert des compétences entre les deux contextes de conception. Nous pensons ainsi que le passage d'un environnement de conception site-central vers un environnement client serveur va conduire…:

Hypothèse 1 :

…à une refonte du sous-modèle d'action, c'est-à-dire à une reconfiguration du mode de gestion et de régulation de l'activité selon les caractéristiques du nouvel environnement cible. Ainsi, la présence d'un canevas de conception hautement prescriptif dans l'environnement site-central obligerait l'informaticien à planifier et à hiérarchiser son activité d'après les objectifs et les contraintes de ce guide.

En revanche, dans l'environnement client serveur, à cause de l'absence de telles structures prescriptives, on s'attend à ce que l'informaticien développe un mode

d'organisation de l'activité plus souple qui autorise et favorise la remise en cause opportuniste de sa tâche.

En conséquence, on supposera que, confronté à l'apprentissage de l'environnement client serveur, l'informaticien spécialisé sur le site-central va spontanément transférer son modèle d'action : il continuera à organiser son activité de manière hiérarchisée alors que le nouveau contexte de conception exigera plutôt une démarche circonstancielle. Le transfert sera négatif dès lors qu'il générera une rigidité cognitive incompatible avec la flexibilité opératoire requise dans le nouvel environnement.

Hypothèse 2 :

... à une redéfinition des stratégies du modèle de résolution. Nous pensons en effet que les caractéristiques de la conception sur *l'environnement site-central* induisent un travail de routine qui s'est substitué aux véritables tâches de conception, c'est-à-dire à une activité de résolution de problèmes originaux. En effet, la stabilité de l'environnement technique, la rigidité du canevas de conception ont contribué à faire de la tâche de conception une simple tâche d'exécution faisant appel à des montages tout faits, stockés sur les sources internes (mémoire) ou externes (bibliothèque d'entités réutilisables). En conséquence, on supposera que l'informaticien site-central raisonne de manière analogique et automatique en récupérant des schémas de résolution déjà écrits, et en les appliquant tels quels au problème.

Dans le cadre de la *programmation client serveur*, nous pensons au contraire que le paradigme événementiel ainsi que les caractéristiques fonctionnelles du système (logique asynchrone, langage interprété...) crée un contexte plus innovant pour la conception. Dans ces conditions, les raisonnements mis en œuvre par l'informaticien devraient être de véritables stratégies de résolution de problème. Il ne s'agirait donc plus de faire correspondre des solutions à un problème, mais de construire, pas à pas, de manière exploratoire et itérative, la solution finale.

Nous supposons dès lors que le novice risque de se livrer à des transferts inappropriés de stratégies de résolution, qui le conduiront à commettre des erreurs de conception. Cette persistance d'anciennes structures mentales induira une situation d'incompatibilité cognitive entre la logique de raisonnement de ce débutant et la logique de développement client serveur.

Hypothèse 3 :

... *à une redéfinition du sous-modèle d'interaction* : nous sommes amenés à penser que les modalités d'interaction proposées par les deux environnements de conception *(interface textuelle et commande par codage pour le site-central ; interface graphique et manipulation directe pour le client serveur)* conditionnent non seulement la qualité de la coopération homme-machine, mais aussi son engagement dans la conception.

Ainsi dans le cadre de l'interaction avec *l'environnement site-central*, on pourra supposer que cette collaboration sera faible à cause des efforts cognitifs que doit consentir l'informaticien pour transformer ces intentions d'action en commandes de programmation.

En revanche, *en environnement client serveur*, la mise à disposition d'une interface très intuitive (menus déroulants, icônes) et facilement utilisable (manipulation directe) favorise le rapprochement des représentations homme-machine, et accentue du coup leur coopération.

Ce qui nous conduit à supposer que l'engagement de l'informaticien sera plus important dans la conception client serveur parce que l'interaction homme-machine est meilleure. En revanche, son engagement sera plus faible en conception site-central car sa logique d'utilisation est plus éloignée de la logique de fonctionnement du système.

En ce concerne l'informaticien novice, celui-ci se donne des tâches de conception à réaliser avec ces dispositifs. Mais se donner de tels objectifs nécessite principalement une bonne connaissance des commandes d'action et des possibilités du système. On peut alors penser que le transfert inapproprié de certaines structures de connaissances dans le nouveau contexte d'interaction homme-machine conduira à de mauvaises interprétations de l'état du système, ou à la définition d'objectifs et de conduites d'interaction impossibles à satisfaire.

Hypothèse 4 :

... *à une redéfinition du sous-modèle de référence* : les schémas de connaissances qui servent de référence pour la conception des composants applicatifs (programmes, interfaces graphiques...) sont étroitement associés aux environnements de conception. Ainsi, dans *l'environnement site-central*, on supposera que ces modèles de référence

sont stables en raison principalement du carcan technique et de l'assujettissement de l'informaticien au système.

A l'inverse, on pronostiquera des modèles évolutifs *en client serveur*, élaborés à partir de données fonctionnelles et graphiques externes à l'environnement. Le système étant dépourvu de sources d'inspiration, l'informaticien ira les rechercher à l'extérieur, sur d'autres logiciels. Son modèle de référence s'enrichira à chaque nouvelle découverte.

En changeant d'environnement informatique, on pose l'hypothèse que l'informaticien novice va conserver certains de ses anciens modèles de référence site-central pour la conception des interfaces en client serveur. Mais, ceux-ci seront inadaptés par rapport aux exigences graphiques et fonctionnelles de la nouvelle situation de conception, et conduiront inévitablement à des erreurs de maquettage.

1.2 DÉMARCHE GÉNÉRALE D'INTERVENTION

Avant de décrire la méthode qui a été employée pour recueillir les données, il convient de situer le contexte de l'intervention.

D'abord, nous préciserons la situation générale du service ou s'est déroulé la recherche. Ensuite nous justifierons le choix de l'étape de maquettage comme objet d'étude. Enfin, nous déterminerons les critères retenus pour délimiter le cadre d'observation.

1.2.1 La situation du service où s'est déroulée l'étude

a) Son organisation

Notre recherche s'est réalisée au sein du service DABFI de l'entreprise Informatique-CDC[29]. Deux raisons à ce choix : la première est que ce service commençait la migration technologique de ses systèmes d'information lorsque débuta notre étude. La seconde est qu'il était représentatif des autres secteurs de l'entreprise par son personnel et les ressources techniques utilisées.

DABFI s'occupe de l'informatique de la Direction des Affaires Bancaires et Financières (DABF) de la Caisse des Dépôts et compte près de 300 personnes dispersées sur 3 sites géographiques différents.

[29] Cf. Annexe II

Regroupés au sein de 6 secteurs (de DABFI-1 à DABFI-7), ces informaticiens sont chargés de développer et de maintenir les applications informatiques dans les grands domaines d'activité de son client : la banque des dépôts, les investissements, les banques de marchés et enfin les métiers de dépositaire et de gestionnaire de fonds. Deux autres secteurs transversaux complètent l'organisation de DABFI : l'un s'occupe de méthodes, d'organisation et des architectures réseaux (Dabfi-0 créé pour accompagner et gérer le changement technique), l'autre d'exploitation et de gestion informatique (Dabfi-7).

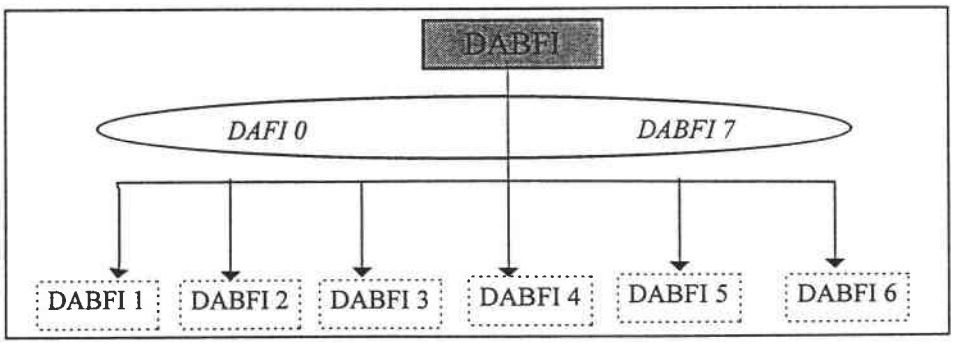


Figure 21
Organigramme fonctionnel de DABFI

b) ***Les raisons de la migration technologique de DABFI***

Ces changements techniques s'inscrivent dans un schéma directeur entamé depuis la fin de l'année 1994. Quatre grands constats ont initié cette migration :

- un système site-central vieillissant et coûteux ne permettant plus de répondre aux besoins des clients ;
- l'évolution technologique du marché qui se caractérise par la percée des micros, les progrès des réseaux, et une offre plus vaste des fournisseurs dans l'architecture client serveur ;
- un niveau d'activité en forte croissante pour la Direction des Affaires Bancaires et Financières, travaillant dans un monde fortement concurrentiel et exigeant des applications toujours plus performantes et conviviales ;
- enfin, un fonctionnement en vase clos qui a conduit à rompre avec le site-central et à s'ouvrir au monde extérieur par le client serveur (tant au niveau conceptuel que fonctionnel).

c) Les systèmes informatiques utilisés dans le service

Les transferts d'apprentissage, dont nous souhaitons étudier les mécanismes, s'inscrivent dans le cadre du passage d'un environnement site-central vers un environnement client serveur, et respectivement de Pacbase vers NSDK. Le premier fait partie de la classe des Ateliers de Génie Logiciel[30] (AGL), le second appartient à la catégorie des Langage de quatrième génération (L4G). Chaque dispositif possède ses propres caractéristiques (techniques, fonctionnelles, logiques et graphiques) que nous serons amenés à détailler dans la description du système homme-machine.

Précisons enfin que, par commodité de langage, nous utiliserons indifféremment les termes "*d'environnement site-central*" ou de "*système Pacbase*" pour désigner l'environnement de conception source ; de même que "*l'environnement client serveur*" et le "*système NSDK*" indiqueront l'environnement cible vers lequel migre l'informaticien.

d) Les caractéristiques de la population de ce service

Le personnel de DABFI est plutôt jeune (70 % des gens ont entre 23 et 35 ans), masculin (65%) et de haut niveau technique (Bac + 4 à Bac + 8). Quatre types de population se distinguent dans ce service :

- Les derniers embauchés (depuis les années 90) viennent d'écoles d'ingénieur ou de formations universitaires en informatique (près de 55%). Ils détiennent une très bonne culture technique sur l'ensemble des techniques de développement. Toutefois, la seule véritable expérience professionnelle dont ils peuvent se prévaloir reste le site-central puisqu'ils ont été recrutés immédiatement après la fin de leurs études. Néanmoins, ils disposent d'un réel potentiel d'adaptation aux technologies client serveur, et abordent le changement avec sérénité.
- Les plus anciens (30 %) ont été engagés dans les années 80, dans un contexte de pénurie de main d'œuvre informatique. Non-informaticien pour la plupart (professeur d'EPS, Docteur en géologie, chimiste...), ils ont appris la programmation dans le cadre des formations dispensées en interne

[30] les caractéristiques de ces deux groupes d'outils ont déjà été largement exposées dans le cadre de l'approche technique de la conception.

par Informatique-CDC. Ils sont donc hyper-spécialisés en développement site central. C'est d'ailleurs cette population qui présente les résistances les plus importantes au changement.

- Un autre profil (15%), comprend des ingénieurs de très haut niveau technique (grandes écoles, doctorat) qui programment déjà dans une architecture client serveur. Ils utilisent cependant des outils différents de ceux préconisés par le schéma directeur.
- Une dernière catégorie de développeurs complète ce panel ; ce sont des prestataires issus de sociétés de services extérieures. Leur rôle consiste d'une part, à aider l'informaticien novice à programmer en client serveur (par des transferts de compétences) et, d'autre part, à soutenir l'activité pour conserver un niveau de productivité acceptable durant cette transition technique. On compte environ un prestataire extérieur pour deux à trois informaticiens internes.

1.2.2 L'étape de maquettage comme objet d'étude

Notre étude se focalise sur la conception de maquette d'application informatique.

a) Définition

Le maquettage est une des étapes du processus de développement dont le but est la génération d'une maquette. Une maquette est un ensemble d'objets graphiques organisés pour donner une image fidèle de l'écran, tel qu'il sera visible par l'utilisateur du futur logiciel. Les différentes fonctionnalités de la future application ne sont pas implémentées, mais les effets qu'auront leur manipulation sur les autres objets de l'interface sont simulés.

b) Justification du choix de l'étape de maquettage

Quatre raisons expliquent le choix de cette étape pour notre étude:

1. La première est que cette étape représente, selon les informaticiens, la phase la plus innovante mais aussi la plus délicate à maîtriser dans la nouvelle activité de conception client serveur. En effet, par ses techniques, ses méthodes et sa logique particulière de conception (logique événementielle, interface graphique), cette procédure marque une rupture complète par rapport au site-

central. C'est une manière résolument différente de concevoir, de se représenter les problèmes et de formuler les solutions.

2. la deuxième raison est que l'interface d'une application constitue en quelque sorte la vitrine commerciale d'un projet. Elle représente le premier point de visibilité du logiciel que le client va juger. Cette évaluation déterminera la poursuite ou l'arrêt du projet. C'est d'ailleurs pour cette raison que les services informatiques accordent une attention toute particulière à cette tâche.

3. La troisième raison est que le maquettage est l'une des rares phases du projet où l'utilisateur a la possibilité de s'exprimer directement sur le développement. Il peut formuler de nouvelles exigences de conception, obliger l'informaticien à modifier certaines écrans, etc.

4. Une dernière raison réside dans le fait que les études qui ont été menées en psychologie ergonomique de la programmation se sont très peu intéressées à cet aspect de la conception. Il n'existe pas, à notre connaissance, de recherches ayant porté exclusivement sur la tâche de maquettage. Ces études se sont davantage intéressées aux étapes de réalisation, de compréhension et de correction des programmes.

1.2.3 Formalisation du cadre d'observation

Pour être en mesure de pouvoir comparer les différentes conduites de conception des trois échantillons retenus, nous avons tenu à déterminer précisément quelles étaient les séquences de maquettage à retenir pour l'observation et l'analyse.

Ainsi, cette tâche se déroule après la phase de spécification (*phase d'analyse détaillée*) et avant la phase de programmation de l'application. Elle débute par une demande de maquettage (émanante du responsable de projet) et se termine par la validation des écrans par l'utilisateur et par le chef de projet. Ces maquettes sont réalisées pour des applications bancaires, selon un cahier des charges défini à l'avance par le comité de pilotage du projet (chef de projets et utilisateurs).

Nous avons également tenu à conserver une dimension homogène entre les différents projets analysés ; à savoir une maquette avec une dizaine d'écrans pour les développements Pacbase, une interface de 4 à 5 fenêtres et avec une vingtaine de fonctions de navigation pour NSDK. Ces deux types de projet réclamaient en moyenne

1 à 2 jours/homme de développement. Nous avons veillé à ce que la taille de l'équipe soit équivalente entre les situations de développement, soit 3 à 4 personnes par projet. Enfin, le niveau de complexité des projets a été supervisé par deux experts "chef de projet" (l'un Pacbase, l'autre NSDK). Ils nous ont orientés vers des développements qui présentaient des difficultés équivalentes et tout à fait admissibles pour des débutants en conception client serveur.

1.2.4 Démarche méthodologique

a) Présentation de la méthode

Notre recherche porte sur une situation réelle et complexe. Elle vise à appréhender l'activité de conception comme une activité située, c'est-à-dire comme une activité étudiée dans son contexte effectif de production. C'est pourquoi la démarche que nous proposons va d'abord définir le contexte de la conception avant d'analyser les processus cognitifs qui y sont déployés.

1. Description du système homme-machine

Nous allons tout d'abord réaliser une description du "*système homme-machine*"[31] relatif aux deux environnements. Cette partie aura une visée essentiellement exploratoire et qualitative. Elle ne prétend pas à l'analyse détaillée des processus cognitifs, mais aura pour but de disposer d'une monographie des situations de maquettage source et cible. Dans cette perspective, quatre dimensions seront dépeintes :

i. *la dimension humaine*, c'est-à-dire les caractéristiques de la population confrontée aux changements techniques. Nous ne reviendrons pas sur ces aspects puisqu'ils ont largement été exposés dans la présentation du personnel de DABFI (Cf. Partie 1.2.1.) ;

ii. *la dimension technique*, c'est-à-dire la description des caractéristiques techniques et fonctionnelles de chaque environnement de conception ;

iii. *une dimension organisationnelle*, c'est-à-dire la présentation de l'activité de maquettage en œuvre dans chaque environnement de conception ;

31 Le système homme-machine correspond à *« une organisation dont les composantes sont des hommes et des machines, reliés par un réseau de communication et travaillant ensemble pour atteindre un but commun »* compte tenu des contraintes d'un environnement donné (Kennedy cité par Sperandio, 1988, p 13)

iv. Enfin, l'analyse *des contraintes* exercées par l'environnement de conception et des *astreintes* ressenties par l'utilisateur viendra compléter cette description. Nous tenterons d'estimer le coût que peut représenter l'utilisation du nouveau dispositif sur l'informaticien.

2. *Analyse des processus cognitifs de conception*

Dans un second temps, nous déterminerons les processus cognitifs de conception et identifierons les transferts d'apprentissage effectués par les débutants.

Pour analyser les raisonnements de conception, on se basera respectivement sur les conduites des experts Pacbase (informaticiens maisons) et de celles des experts NSDK (prestataires). Ces professionnels manifestent en effet des conduites qui sont représentatives des environnements techniques dans lesquels ils oeuvrent. En outre, l'étude de leurs stratégies et de leurs représentations mettra en évidence des questions nouvelles, qui ne se posent pas encore chez le débutant.

Pour identifier la nature des transferts d'apprentissage, nous déterminons les modèles mentaux de l'informaticien débutant en NSDK et les comparerons à ceux des deux autres catégories d'experts, de telle sorte que (Cf. Figure 22) :

i. les modèles mentaux des experts Pacbase renseignent sur les compétences acquises dans le domaine source ;
ii. les modèles mentaux des experts NSDK informent sur les compétences requises pour programmer dans le domaine cible ;
iii. l'analyse des modèles mentaux des novices NSDK signalent les compétences effectivement transférées vers le client serveur.

En comparant l'ensemble, on sera alors en mesure de déterminer quelles sont les compétences que le novice a récupérées du domaine source (en les confrontant à celles des experts Pacbase), et si elles sont appropriées ou non au domaine cible (d'après celles des experts NSDK).

C'est l'application du modèle d'analyse qui fournira la grille de lecture de ces processus, en distinguant notamment, pour chaque échantillon d'informaticiens :

i. *le sous-modèle d'action*, qui analyse les modes d'organisation et de régulation de l'activité ;

ii. *le sous-modèle de résolution,* qui dégage les stratégies de résolution en jeu ;

iii. *le sous-modèle d'interaction,* qui identifie les connaissances requises pour interagir avec le système et qui souligne le niveau d'engagement ou de retrait dans la conception ;

iv. *le sous-modèle de référence,* qui décrit les exemplaires typiques mémorisés dont s'inspirent les informaticiens pour concevoir l'interface.

Au final, les données recueillies (experts Vs novices) permettront d'accéder à une meilleure compréhension des processus de conception ; en mettant en évidence les différences de contenu des connaissances manipulées, en en indiquant aussi la manière dont celles-ci sont (ré)employées.

<table>
<tr><th></th><th>Experts Pacbase</th><th>Experts NSDK</th><th>Novices NSDK</th></tr>
<tr><td>Modèle d'action</td><td rowspan="4">Compétences acquises dans l'environnement source</td><td rowspan="4">Compétences requises dans l'environnement cible</td><td rowspan="4">Compétences effectivement mises en œuvre par les débutants</td></tr>
<tr><td>Modèle de résolution</td></tr>
<tr><td>Modèle d'interaction</td></tr>
<tr><td>Modèle de référence</td></tr>
</table>

Figure 22
Démarche d'analyse comparative des modèles mentaux de chaque échantillon d'informaticiens

b) Description générale de l'échantillon retenu pour l'étude

Nous présentons ici les caractéristiques générales de l'échantillon utilisé pour l'ensemble de cette recherche. Le nombre exact de personnes impliquées dans chaque étude sera reprécisé dans leur partie respective.

Trois groupes homogènes (par l'âge, l'ancienneté et l'expérience des environnements) ont donc été constitués (Cf. Figure 23) :

- 9 experts en environnement Pacbase
- 8 novices en environnement NSDK (c'est-à-dire débutants), mais expérimentés en conception Pacbase,

- 7 experts en conception NSDK accompagnant les novices dans leur apprentissage de l'environnement. Ces experts sont en fait les prestataires extérieurs dont certains sont dans le service depuis plusieurs mois.

	9 Experts Pacbase	8 Novices NSDK	7 Experts NSDK
Age	32- 35 ans	29-33 ans	30-33 ans
Ancienneté dans l'entreprise	6-10 ans	6-10 ans	3 mois-1 an
Expérience de l'environnement	6-10 ans	1-3 mois	2 -3 ans

Figure 23
Caractéristiques de l'échantillon étudié

c) Techniques de recueil de données et méthode d'analyse des protocoles

1) Les techniques de recueil de données

Ces techniques seront exposées lors de la présentation des différentes analyses effectuées sur le terrain. On trouvera également en annexe[32] une description plus détaillée de ces instruments, ainsi qu'une discussion portant sur la validité des méthodes par verbalisation

2) Méthode d'analyse des protocoles

L'enquête repose sur une méthodologie quantitative et qualitative. Dans ce dernier cas, l'intérêt du matériau collecté ne réside pas seulement dans la dimension de représentativité mais aussi dans l'accès à la compréhension d'un processus complexe : la conception d'une interface graphique dans un environnement informatique. Il convient donc d'exposer rapidement quelle a été notre méthode d'analyse des protocoles recueillis.

Interpréter de tels indicateurs sous l'angle de conduites cognitives nécessite de mettre au point des descripteurs des protocoles verbaux recueillis auprès des sujets. Pour ce faire, nous avons transformé les protocoles bruts en protocoles analysables en les découpant en unités d'expression qui se différencient selon la catégorie d'activité cognitive à laquelle elles font référence –catégories que nous avons répertoriées dans les modèle mentaux–, à savoir :

- la planification de l'activité de conception (opportuniste ou hiérarchisée),

[32] Cf. Annexe III

- les stratégies de résolution (compréhension du problème, démarche ascendante…),
- la nature des interaction homme/machine.
- les connaissances mobilisées.

Puis, pour chaque unité linguistique, nous nous sommes demandés à quel type d'opération cognitive elle pouvait se référer : "*expression d'un but à atteindre*", "*description d'un but intermédiaire de la solution*", "*recours à une procédure d'exécution*", "*explication d'une stratégie de résolution...*"

Si l'on prend par exemple ce cas où un développeur nous déclare, à propos de sa démarche de conception :

«si tu n'as pas une bonne ***visibilité mentale de l'itinéraire*** *qui va t'amener à l'état final, et bien, tu ne t'engages pas»*,

nous interprétons "*bonne visibilité mentale de l'itinéraire…*" comme un indicateur faisant référence à une "planification hiérarchisée" de la résolution. Ce morceau de corpus peut également révéler un prérequis de la conception : maîtriser parfaitement le plan de résolution avant de débuter le développement ("[…]*Tu ne t'engages pas*").

Bien évidemment, cet exemple comme toutes les interprétations que nous exposerons ultérieurement, représentent des conjectures qu'il convient d'étayer et de renforcer par les recueils complémentaires (observation de l'activité) et par des morceaux de corpus prélevés chez d'autres individus (pour tendre vers la généralisation des résultats).

En définitive, cette méthode permet d'obtenir une description des protocoles qui rend compte à la fois des connaissances, des représentations et des stratégies mises en oeuvre par les concepteurs.

2. LA DESCRIPTION DU SYSTÈME HOMME-MACHINE : LE CONTEXTE DE L'ANALYSE

Nous commençons par comparer les caractéristiques techniques de chaque dispositif de conception (site-central Vs client serveur). En fonction des écarts constatés, on indique les aspects techniques pour lesquels l'informaticien novice devrait procéder soit :

- *à un apprentissage spécifique (cas de divergence ou de nouveauté technique) ;*
- *à un réajustement de ses compétences (cas de la similarité technique) ;*
- *à la réutilisation de ses anciennes connaissances site-central (cas de convergence ou de l'analogie technique).*

Ensuite, l'organisation de l'activité de maquettage en œuvre dans chaque situation de conception est présentée. Leur comparaison est l'occasion de déterminer précisément quelles sont les transformations que subit l'activité de maquettage lorsqu'elle est exécutée dans l'environnement site-central, puis en client serveur.

Enfin, une dernière partie répertorie les différentes catégories de contraintes générées par les nouveaux dispositifs informatiques. Cet inventaire nous permet d'évaluer les coûts (astreintes) que représentent ces évolutions techniques pour l'informaticien.

2.1 LES CARACTÉRISTIQUES TECHNIQUES DES ENVIRONNEMENTS DE CONCEPTION ÉTUDIÉS

2.1.1 Méthode employée

a) Méthodes de recueil de données

Sur la base des caractéristiques techniques définies dans le chapitre 1, nous avons dressé quatre grands critères de comparaison entre les environnements de conception source et cible. Il s'agit des caractéristiques : 1) architecturales, 2) fonctionnelles; 3) logiques ; 4) graphiques.

Pour ces différentes analyses, nous nous sommes assurés le concours de deux experts, spécialisés respectivement sur les dispositifs site-central et client serveur. Ils font office de formateurs et d'assistants technique auprès des concepteurs. Nous avons également consulté la documentation technique livrée avec ces outils ainsi que celle, plus spécifique, éditée par les services méthodes (DABFI-O). Nous nous sommes enfin livrés à une analyse ergonomique et fonctionnelle des dispositifs grâce à une grille[33] d'analyse développée sur la base des critères ergonomiques de Bastien et Scapin (1994).

b) Méthode d'analyse des données

Pour chacune des caractéristiques techniques considérées, nous avons cherché à déterminer, avec l'aide des experts, quelles étaient celles qui se révélaient :

- tout à fait inédites dans l'environnement client serveur (*indice de nouveauté*) ;
- similaires aux deux environnements (*indice de similarité*) ;
- identiques aux deux environnements (*indice d'analogie*).

Et, en fonction de l'importance de ces écarts, on a indiqué pour quels aspects l'informaticien devait plutôt :

- développer un apprentissage spécifique (*nouveauté technique*) ;
- procéder à un réajustement de ses compétences (similarité technique) ;
- ou bien, réutiliser directement ses anciennes connaissances du site-central (analogie technique).

[33] Cf. Annexe IV

2.1.2 Les caractéristiques architecturales des environnements de conception étudiés

Elles correspondent à l'organisation générale des systèmes informatiques associant un poste de travail à un environnement plus vaste : gros système (mainframe), terminaux et bases de données (DB2 et DL1) dans le cas du site-central ; postes clients reliés à un serveur et à une base de données Sybase dans le cas des systèmes distribués client serveur.

2.1.2.1 L'architecture centralisée du site-central

L'outil de conception Pacbase fonctionne dans une architecture centralisée, composée d'un terminal passif relié à un site-central. Cette architecture comprend d'un côté des terminaux et de l'autre un gros système central (le mainframe), et des bases de données[34] (DB2 et DL1) reliés entre eux par un réseau. Ce mainframe centralise les programmes composants le coeur fonctionnel de l'application (Cf. Figure 24). Les bases de données alimentent l'application en données d'après les requêtes effectuées sur le terminal (par exemple, extraction de données, mise à jour de données...). Remarquons enfin que l'organisation de cette architecture est totalement transparente pour l'informaticien. Celui-ci n'a pas à se soucier de l'emplacement de telles ou telles bases de données ou de tels ou tels programmes sur le site-central. Tout cela est géré automatiquement par le système.

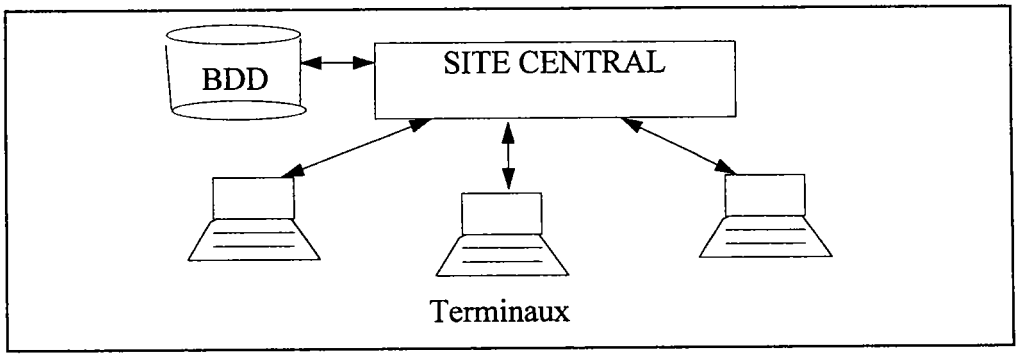


Figure 24
Modèle de l'Architecture site-central

[34] bases de données : BDD

2.1.2.2 L'architecture distribuée du client serveur

Cette architecture client serveur se subdivise en deux parties : une partie "*client*" émet des demandes de service, le "*serveur*" exécute les requêtes émanantes des différents clients. La réelle nouveauté tient au fait que l'informaticien doit dorénavant tenir compte de cette architecture lors de la conception informatique. Ainsi, il doit décider si la gestion de l'interface, des bases de données et des traitements revient prioritairement au poste "client" ou au poste "serveur". De cette répartition fonctionnelle dépendront les performances de l'application finale (débit des données, temps de réponse, vitesse des traitements).

Ce souci permanent d'efficacité est peu présent en site-central puisque toutes les procédures informatiques (traitements, transactions sur les bases de données, présentations graphiques) sont intégralement prises en charge par le mainframe.

En outre, si le site-central est réputé pour sa robustesse et sa sécurité, le client serveur, en revanche, se révèle être une technologie fragile et peu fiable. Il arrive ainsi souvent que des arrêts intempestifs suppriment tout le travail de conception non sauvegardé.

Une dernière particularité de cette architecture est le nouveau Système de Gestion des Bases de Données (SGBD) "*Sybase*" dont le langage de requêtes (par procédures stockées) diffère de celui des bases "DB2" et "DL1" du site-central.

2.1.2.3 Synthèse comparative

Ce tableau rappelle les principaux points de divergence ou de convergence entre les caractéristiques architecturales des deux environnements de conception considérés. Il a été constitué à partir de nos discussion avec les experts techniques (comme tous les tableaux de comparaison qui seront exposés dans cette partie).

CARACTÉRISTIQUES ARCHITECTURALES	ENVIRONNEMENT SITE-CENTRAL	ENVIRONNEMENT CLIENT SERVEUR	INDICE DE NOUVEAUTÉ	INDICE D'ANALOGIE	INDICES DE SIMILARITÉ
Configuration de l'architecture	– Centralisée	– Distribuée	●		
Place de l'architecture dans la démarche conception	– Transparente	– Prépondérante pour les performances de l'application	●		
Bases de données	– DB2 et DL1	– Sybase et procédures stockées	● (au niveau du langage et de la logique)		● (au niveau du principe fonctionnel)
Fiabilité de l'architecture	– Architecture fiable et sécurisée	– Architecture peu fiable et peu sécurisée	●		

Figure 25

Tableau de comparaison des caractéristiques architecturales des environnements de conception (élaboré à partir de l'avis des experts techniques)

a) le premier changement concerne la ***configuration de l'architecture*** qui passe donc d'une structure "centralisée" à une organisation "distribuée". Les particularités techniques du client serveur sont fondamentalement différentes de celles du site-central et induisent de ce fait de nouvelles contraintes de développement. Par exemple, l'informaticien doit déterminer les fonctions des postes client et serveur dans le but d'optimiser les performances de l'application.

b) Une autre particularité touche au ***manque de fiabilité*** de l'architecture client serveur. Les dysfonctionnements du système sont d'ailleurs considérés comme une "régression" technique par l'informaticien dans la mesure où celui-ci doit être beaucoup plus vigilant dans ses procédures de développement et de sauvegarde.

c) On relève enfin que les ***SGBD*** **évoluent** : DB2 et DL1 sont remplacés par Sybase. Si le principe fonctionnel reste identique ("*extraire*", "*modifier*", "*enrichir*" et "*sauvegarder*" des tables de données sont des fonctions communes aux deux systèmes), les commandes et certaines procédures d'utilisation diffèrent cependant (notamment au niveau des procédures stockées[35]).

2.1.3 Les caractéristiques fonctionnelles des environnements de conception étudiés

Ces caractéristiques correspondent aux fonctionnalités de l'environnement qui permettent de programmer et de tester l'interface, de stocker et de rechercher des entités de programmation. Nous allons d'abord énumérer les spécificités fonctionnelles de Pacbase et de NSDK, pour rendre compte ensuite de leurs niveaux de convergence/divergence.

2.1.3.1 Spécificités fonctionnelles de Pacbase

Quatre composantes de l'AGL Pacbase interviennent dans la conception des applications centralisées. Il s'agit :

a) du dictionnaire de données ;

b) des squelettes de programmation "TP" et "Batch" ;

c) des commandes nécessaires pour l'utilisation de Pacbase ;

d) d'un langage compilé.

[35] Une procédure stockée est une procédure d'accès automatisée aux bases de données .

a) Le dictionnaire d'entités

Ce dictionnaire stocke des entités de conception standards qui seront réutilisées lors du développement de l'application. Des blocs de programme, des exemplaires d'écrans préfabriqués, etc. composent le gros de ces entités informatiques.

b) Les squelettes de programmation

Pacbase propose deux squelettes de programmation qui vont guider et surtout encadrer l'informaticien dans le développement de l'application. Ces canevas remplissent deux fonctions de développement:

1. un squelette TP pour des traitements transactionnels[36] instantanés ;
2. un squelette Batch pour les traitements différés.

Ces deux canevas assurent la standardisation des programmes entre tous les projets de développement. Pour concevoir, l'informaticien s'insère au niveau des fonctions du squelette[37] et saisit les instructions de programmation par l'intermédiaire d'un langage spécifique : le langage "Pac" (Cf. Figure 26). Une fois cette opération terminée, Pacbase se charge de générer automatiquement le code source cobol[38] (Cf.Figure 27).

[36] Il y a une mise à jour des bases de données et des écrans directement après que l'utilisateur ait agit sur l'application

[37] La structure logique de ce programme TP est présenté en "*Annexe V*".

[38] Le langage Pac comme le code cobol généré appartiennent à la catégorie des paradigmes procéduraux.

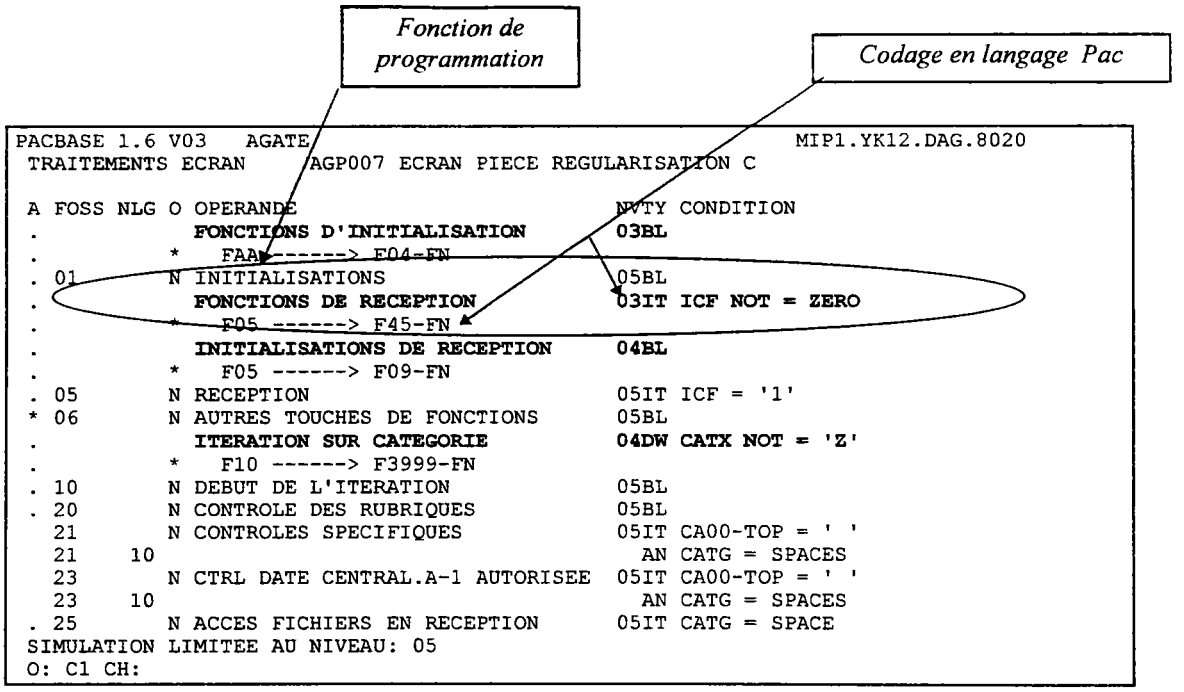


Figure 26
Ecran de saisie de Pacbase :
exemple de développement avec le squelette de programmation TP

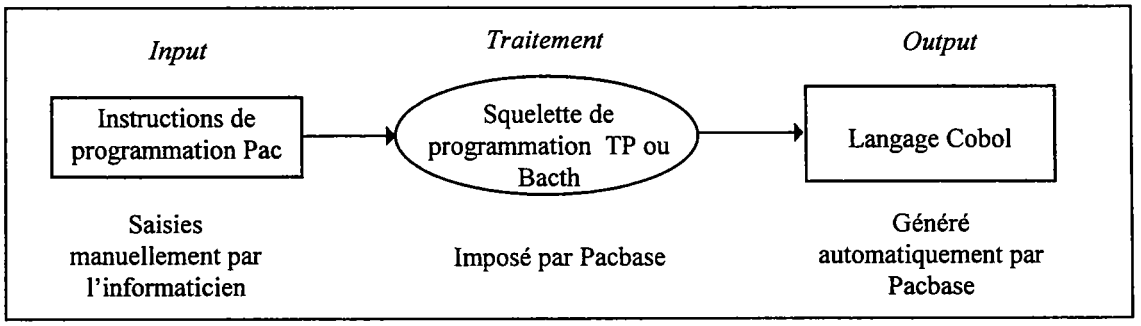


Figure 27
Schéma de programmation dans Pacbase

c) Conception et navigation dans Pacbase.

La conception avec Pacbase : Pacbase requiert l'utilisation d'un nombre important de codes et de commandes pour développer. On en a comptabilisé plus de 300 dont la syntaxe se révèle être absconse et compliquée. Ces commandes ont en outre la possibilité de se combiner pour en former d'autres, aux fonctionnalités plus larges. Par exemple, la commande *"E...XP.....P..."* est formée de plusieurs commandes signifiant *"Utilisations de la rubrique (code E)... dans la programme (code P) ... à partir des lignes P"*

La navigation dans Pacbase : Pour naviguer dans Pacbase, l'informaticien dispose de deux techniques (Cf. Figure 28) :

- soit il appelle un "*menu général*" qui écrase l'écran sur lequel il se trouve. Il sélectionne alors la commande à réaliser parmi la liste de propositions ;
- soit il tape directement une ligne de commande dans une zone prédéfinie de l'écran.

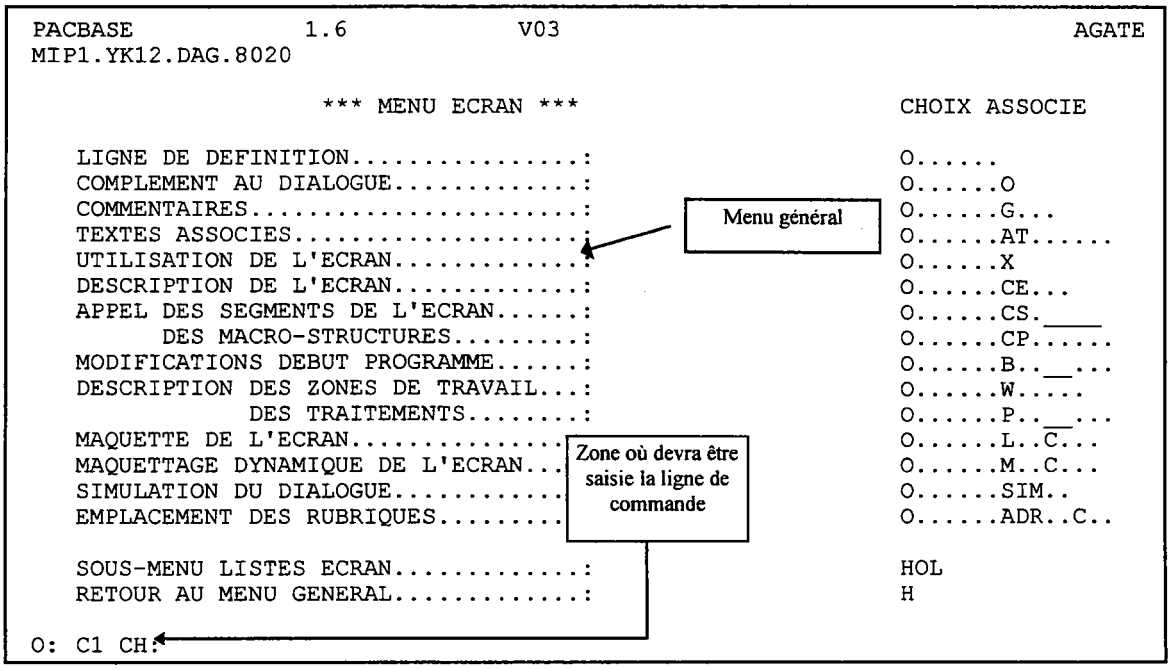


Figure 28
Ecran de commandes pour accéder aux écrans de Pacbase

d) Un code compilé

Le langage Pac est un code compilé qui nécessite une opération de compilation avant d'être exécuté. Cela consiste à transformer le langage source en langage machine. Sans compilation, il n'est pas possible de tester le programme ou de simuler la maquette élaborée. Mais chaque compilation génère un coût machine appelé CPU. Par mesure d'économie, les responsables ont donc décidé de restreindre la consommation de chaque informaticien. Mais l'instauration de ces "quotas" a induit, insidieusement, des "effets contre-productifs" : les informaticiens minimisent le recours aux tests techniques pour ne pas dépasser la limite de CPU autorisée. Seulement, les erreurs de programmation qui auraient pu être détectées par le biais de ces simulations, le seront bien plus tard, lors des tests finaux. Au total, leur correction se révélera plus fastidieuse et surtout plus onéreuse.

2.1.3.2 Spécificités fonctionnelles de NSDK

NSDK est un environnement de conception intégré qui comporte, outre les outils classiques de développement (éditeur de programmes, déboggeur), un générateur d'Interface Homme-Machine (IHM), des librairies et un langage interprété. Nous allons rapidement les décrire.

a) Un générateur d'IHM

Ce générateur sert à l'édition graphique des composants de l'interface (menus, fenêtres, icônes, boutons...) ainsi qu'à la spécification de leurs comportements (Fermeture d'une fenêtre, changement d'état d'un menu...). Trois grandes opérations de maquettage vont être réalisées par l'intermédiaire de cet éditeur :

- *la composition des fenêtres* par le positionnement d'objets graphiques (appelés aussi "contrôles") ;
- *la gestion des fenêtres* : on y définit l'affichage des fenêtres, l'ouverture, la fermeture, la superposition, le déplacement et le passage au premier ou en arrière plan ;
- *la gestion des événements* : l'informaticien gère toutes les entrées de l'utilisateur (déplacement souris, appui ou relâchement d'un bouton, appui d'une touche clavier...) et détermine le comportement que l'application doit adopter en conséquence. Par exemple, la sélection du bouton droit de la souris va générer un événement qui déclenchera l'ouverture d'une menu contextuel.

Outre cet éditeur graphique, NSDK se caractérise aussi par son code interprété et les nombreuses librairies mises à la disposition de l'informaticien

b) Un code interprété

Le programmation de l'interface se fait par un langage spécifique à l'outil –NCL– qui génère un langage machine le "C" (Cf.Figure 29). NCL est un langage interprété. Cette particularité autorise l'informaticien à tester directement les composants développés sans passer par la compilation, et surtout sans que cela ne génère un surcoût machine (CPU). En somme, cette propriété du langage favorise la mise au point des applications informatiques dans la mesure où les modifications introduites dans le code pourront être immédiatement simulées et évaluées.

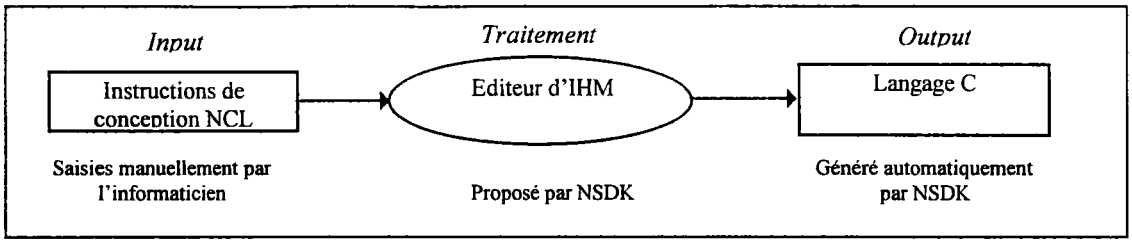


Figure 29
Schéma de conception dans NSDK

c) Les librairies

Les librairies regroupent des fonctions informatiques préfabriquées que les informaticiens peuvent réutiliser lors de la conception. On a comptabilisé huit librairies spécialisées sur des grands domaines de développement : instructions de programmation pour la programmation et le maquettage, objets graphiques pour le maquettage, procédures stockées pour la gestion des bases de données.... Toutefois, à cause de la relative jeunesse de l'environnement client serveur, la plupart de ces librairies n'offrent que très peu de fonctions réutilisables. Les informaticiens sont alors obligés de créer ces composants de toute pièce, ou bien, de solliciter l'intervention du secteur DABFI O pour les développer. Certains développeurs s'adressent même directement aux autres équipes de conception pour leur demander les fonctions recherchées.

2.1.3.3 Synthèse comparative

Le tableau ci dessous rappelle les principaux changements fonctionnels intervenus lors du passage de Pacbase vers NSDK. Les niveaux de convergence ou de divergence fonctionnelle sont signalés par les indices de "nouveauté", "d'analogie" et de "similarité".

CARACTÉRISTIQUES FONCTIONNELLES	ENVIRONNEMENT SITE-CENTRAL	ENVIRONNEMENT CLIENT SERVEUR	INDICE DE NOUVEAUTÉ	INDICE D'ANALOGIE	INDICE DE SIMILARITÉ
Paradigme du conception	– Procédural (Pac et Cobol)	– Orienté objet graphique	●		
Propriété du langage	– Compilé (Pac)	– Interprétée (NCL)	●		
Dispositifs de stockage des informations	– Un dictionnaire commun pour toutes les entités de programmation	– Plusieurs librairies spécialisées par fonctions réutilisables (peu disponibles et peu fonctionnelles)			●
Canevas de conception de l'application	– Squelettes de programmation (TP, Batch), macro-instructions et codification (300 codes) – Très prescriptif	– Générateur d'interfaces graphiques (souris, manipulation directe...). – Souple, flexible et non déterministe	●		
Nature des instructions de programmation	– macro-instructions de Pacbase et de l'entreprise	– Macro-instructions de NSDK et de l'entreprise			●
Nature du code généré	– Langage procédural "Cobol" (Pac→cobol).	– Langage procédural "C" (NCL→C)		●	

Figure 30
*Tableau comparatif des attributs **fonctionnels** des environnements de conception (élaboré à partir de l'avis des experts techniques)*

a) Un premier changement concerne ***l'évolution des paradigmes de programmation :*** l'informaticien passe ainsi d'un paradigme procédural de programmation (on programme des instructions) vers un paradigme orienté "*objet graphique*" de conception (on déplace des objets graphiques). Cette évolution nécessite l'adoption de nouveaux modes opératoires (savoir utiliser et déplacer une souris) et l'acquisition de modèles de développement plus conceptuels (ne plus se représenter une application en termes de procédures informatiques, mais en termes d'objets virtuels).

En outre, les langages de conception utilisés (Pac et NCL) s'opposent par leur syntaxe, leur règle d'écriture et leur code sémantique. En revanche, les langages générés (Cobol et C) sont tous deux de nature procédurale. Une analogie peut donc être envisagée entre ces deux paradigmes, et la compréhension du langage C pourrait être ainsi facilitée par la connaissance du langage cobol.

b) Autre changement observé, la disparition des ***canevas de conception*** : Sur Pacbase, les squelettes TP et Batch déterminent fortement l'action de l'informaticien, tandis que l'éditeur graphique de NSDK n'impose rien et laisse au sujet le soin d'organiser librement son activité. Le développeur ne se trouve donc plus assujetti à des canevas prescriptifs.

c) De même, la ***propriété interprétée*** du langage NCL marque une évolution importante dans les habitudes de conception de l'informaticien puisque celui-ci a la possibilité de tester et de simuler, régulièrement et sans limite, l'état d'avancement de son projet de maquettage. Cette nouvelle fonctionnalité conduit à une démarche de conception itérative pas ajustements progressifs, du type :
"*développement → test (par simulation) → correction*".

d) Enfin, les ***dispositifs de stockage*** de chaque environnement présentent des caractéristiques fonctionnelles très similaires, malgré le fait que l'on trouve un dictionnaire unique sur Pacbase et plusieurs librairies spécialisées sur NSDK. Ces deux bases font office de réserve de productivité mobilisable. Elles jouent un rôle important parce qu'elles constituent un ensemble d'informations stockées sous des formes diverses et dont se servent les informaticiens pour résoudre à leur façon les problèmes qui se posent à eux. Ces réserves (sorte d'extension matérielle de la mémoire cognitive et collective) leur permettent d'affronter plus sereinement les aléas et les incertitudes de la conception, et d'en apprivoiser les formes.
Dans ce cas, les réapprentissage porteront plutôt sur le nom des entités proposées par les librairies de NSDK, sur leur fonction respective, sur leur localisation dans les bases et sur la procédure pour les rechercher et les extraire.

2.1.4 Les caractéristiques graphiques des environnements de conception étudiés

Ces caractéristiques font référence aux modalités d'interaction proposées par chaque environnement de développement pour concevoir la maquette : interface textuelle de programmation sur Pacbase, interface à base d'objets graphiques et de manipulation directe sur NSDK.

2.1.4.1 L'interface alphanumérique de Pacbase

Le terminal de travail comporte une interface en mode textuel. La manipulation de Pacbase passe par la saisie d'une chaîne d'instructions complexes qui a déjà été exposée dans la partie concernant les caractéristiques fonctionnelles de l'environnement (Cf. Figure 26). La conception de la maquette se fait au moyen d'une grille de saisie (ou grille de maquettage). L'informaticien paramètre les écrans en codifiant les zones spécifiques de la grille : il y indique le nom des rubriques utilisées,

leur libellé, leur emplacement sur l'écran, leur couleur, etc. (Cf. Figure 31). Cette technique de maquettage est appelée "-CE[39]". Le mutifenêtrage étant impossible, la simulation conduira à l'apparition de l'écran maquetté et à la disparition de sa grille de paramétrage (Cf. Figure 32).

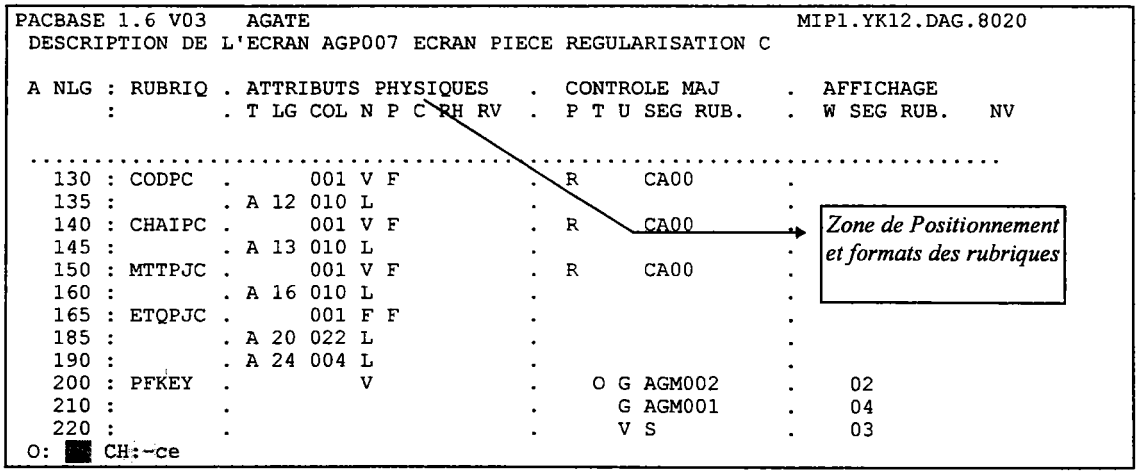


Figure 31
Grille de maquettage "-CE" : description de l'écran de la maquette

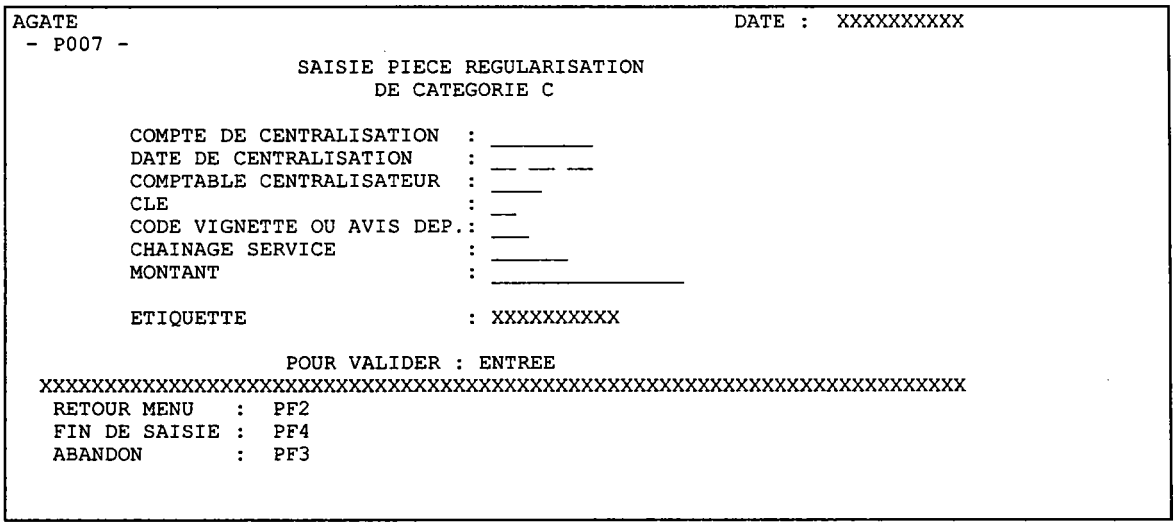
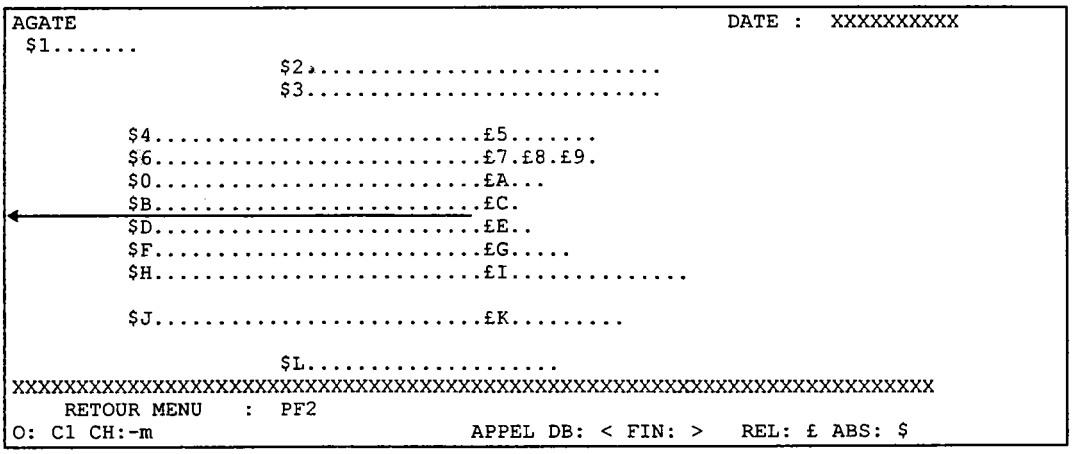


Figure 32
Exemple d'écran maquetté et simulé

[39] en référence à la commande permettant d'accéder à la grille de maquettage.

Il existe aussi une autre technique de maquettage appelée "**Maquettage dynamique**" (-M). Elle consiste à appeler et à placer directement les rubriques sur un écran vierge par une série de codes informatiques (Cf.Figure 33). Le principal attrait de cette méthode est de fournir une image instantanée de la maquette conçue, et ce, sans compilation. L'informaticien peut réaliser ses corrections directement à l'écran, sans passer par la grille de maquettage.

Codifications permettant d'appeler, de placer, et de paramétrer directement les rubriques sur un écran vierge


AGATE DATE : XXXXXXXXXX
$1.....
$2.....
$3.....
$4.....£5.....
$6.....£7.£8.£9.
$0.....£A...
$B.....£C.
$D.....£E..
$F.....£G.....
$H.....£I.....
$J.....£K.....
$L.....
XXXXXXXXX

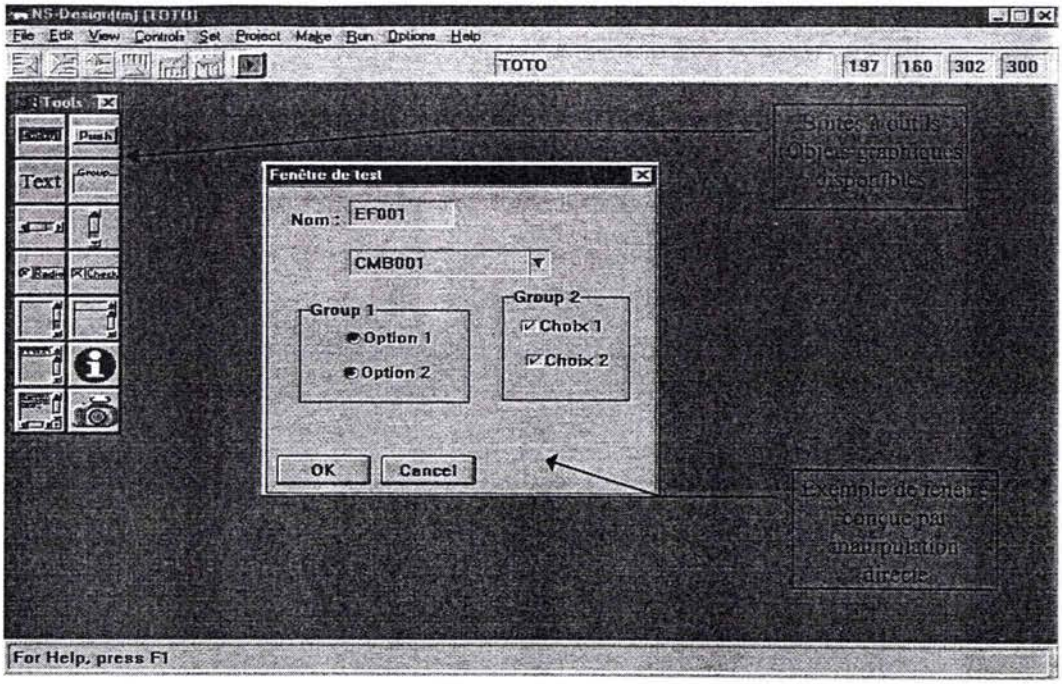


Figure 34
Plan de travail type de l'éditeur graphique de NSDK

a) La conception Graphique

Grâce à l'interface graphique de conception, l'informaticien n'a plus à se remémorer et à saisir des codes abscons pour définir l'aspect d'une fenêtre. Il développe de nouveaux modes opératoires basés sur le "drag and drop", c'est-à-dire la manipulation directe à l'aide de la souris. Il doit également savoir paramétrer les objets graphiques (ou contrôles) en spécifiant leur graphisme, leur géométrie et leur comportement :

- *le graphisme* : on regroupe en fait sous ce terme des attributs graphiques tels que le type et l'épaisseur de traits, les motifs, les couleurs, les polices des boutons, etc ;
- *la géométrie* : c'est l'ensemble des attributs définissant les dimensions physiques du contrôle ainsi que sa position : soit à l'intérieur même d'une fenêtre (positionnement absolu), soit par rapport à d'autre contrôles (positionnement relatif) ;
- *Le comportement* : cela fait référence aux différentes conduites que doit adopter la maquette lorsqu'on sélectionne une de ses commandes (ouverture d'une fenêtre ou lancement d'une fonction de calcul). La définition de ces comportements nécessite une phase de programmation que sera exposée plus loin.

Cet éditeur graphique permet à l'informaticien d'évaluer instantanément l'évolution de la maquette au fur et à mesure qu'il positionne les objets graphiques sur l'écran. De plus, Le multifenêtrage permet d'afficher dans une fenêtre voisine les composants de la maquette, de simuler leur comportement et de les corriger directement sur la fenêtre principal de l'éditeur graphique (Cf. *Figure 34* ; *Figure 35*).

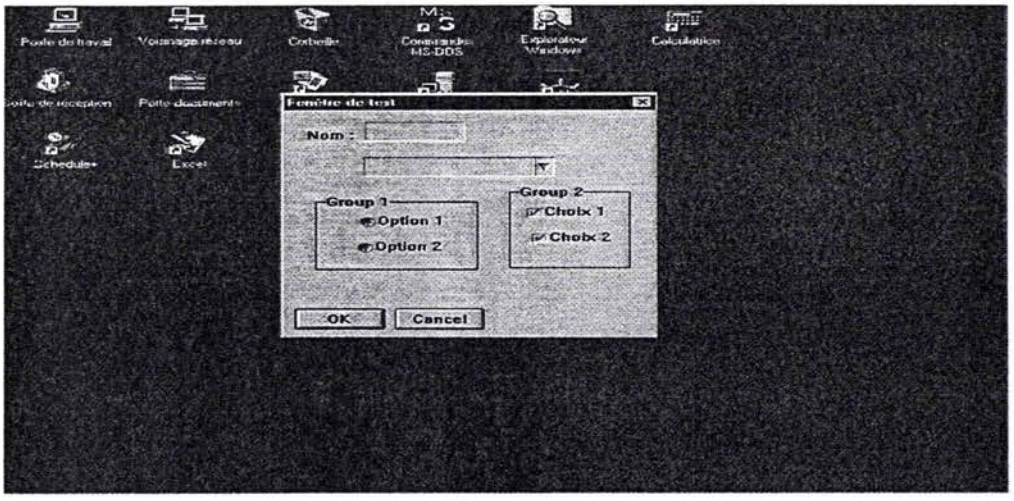


Figure 35
Maquette simulée

b) La navigation dans NSDK

Un menu, des boutons, des icônes et des boîtes de dialogue fournissent une assistance continue à l'informaticien pour naviguer dans NSDK (Cf. *Figure 37*). Il interagit avec cet environnement en effectuant des choix à l'aide de la souris ou au moyen de raccourcis clavier. Notons aussi qu'il existe une aide en ligne ainsi qu'une "barre d'état" informant l'informaticien sur les actions qu'il est sur le point d'entreprendre, ou qu'il est en train de réaliser.

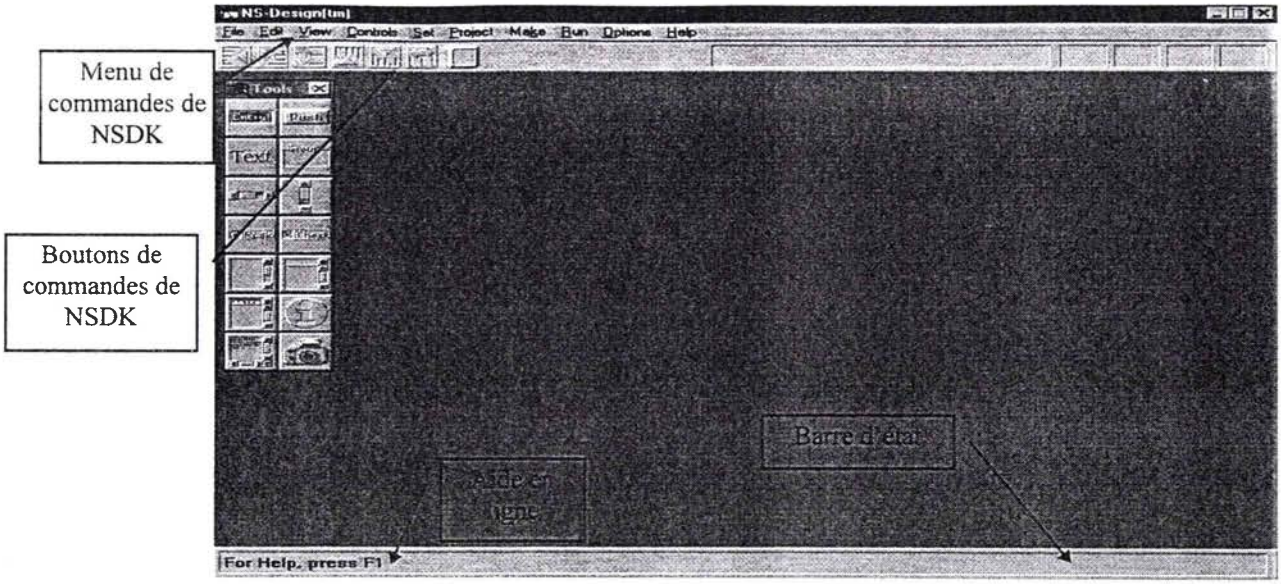


Figure 36
Ecran de maquettage :Zone de travail

c) Définition du comportement de la maquette : la programmation

La programmation des comportements de l'interface se déroule dans une boîte de dialogue où sont spécifiés toutes les fonctions, les événements et les instructions nécessaire à cette tâche. (Cf. *Figure 37*).

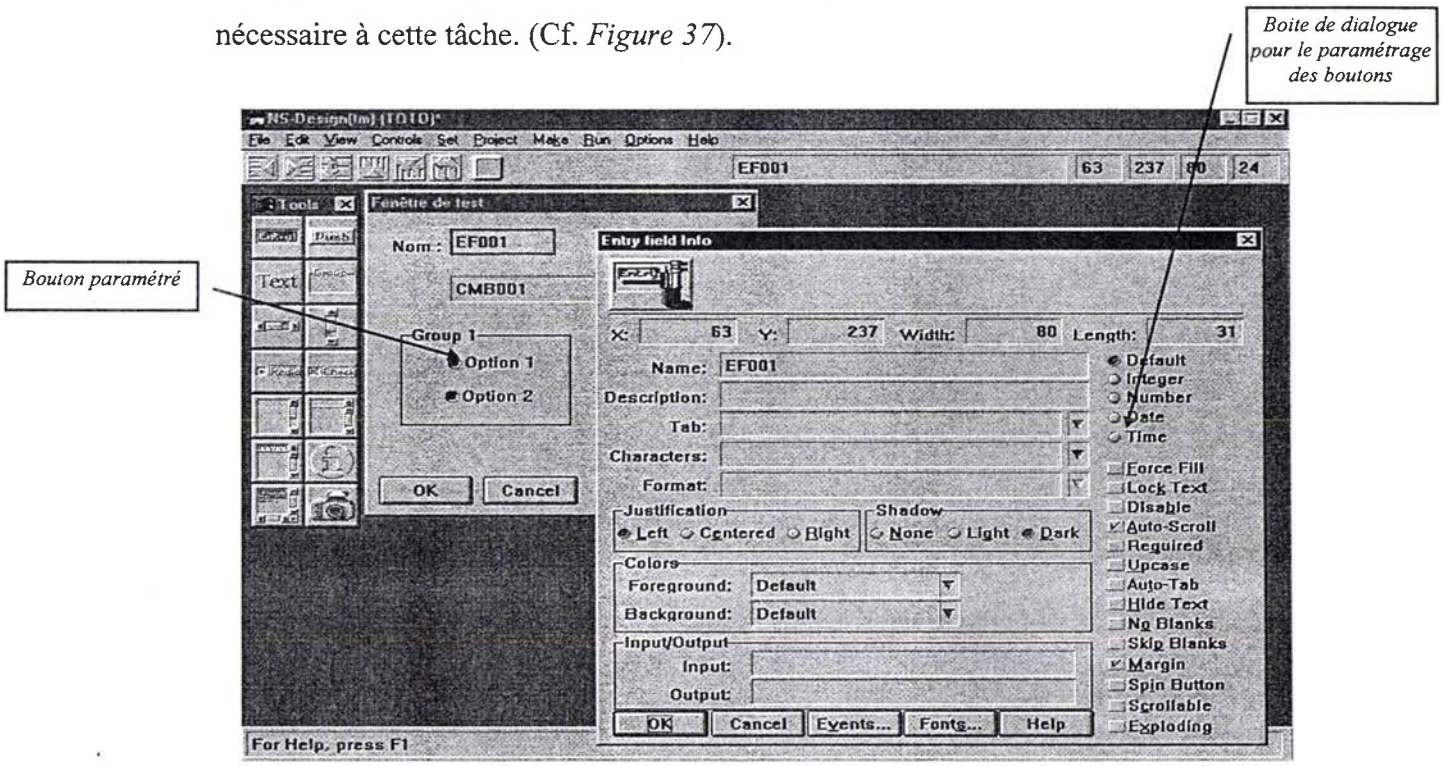


Figure 37
Ecran de maquettage : boîtes de dialogue pour paramétrer les boutons (graphisme, comportement, géométrie)

En somme, en dépit du très grand nombre de commandes à retenir pour composer et programmer la maquette, l'interface graphique de NSDK offre un environnement convivial qui favorise la navigation dans l'outil.

2.1.4.3 Syntèse comparative

Ce tableau met en perspective les aspects graphiques des deux environnements de conception.

CARACTÉRISTIQUES GRAPHIQUES	ENVIRONNEMENT SITE-CENTRAL	ENVIRONNEMENT CLIENT SERVEUR	INDICE DE NOUVEAUTÉ	INDICE D'ANALOGIE	INDICE DE DIMILARITÉ
Nature de l'interface	– Alphanumérique	– Graphique	●		
Mode d'affichage Modalité d'interaction	– Fenêtrage unique – Codage	– Multifenêtrage – Manipulation directe, sélection de boutons, boîtes de dialogues	●		
Modes opératoires de manipulation des environnements de conception	– Par touche-fonction, et saisie d'instructions de commande	– Manipulation directe et sélection dans listes d'items (utilisation de la souris, des menus, des icônes…)	●		
Maquettage	– Par codification sur des grilles de saisie (-CE), – En direct sur des écrans vierges (maquettage dynamique, -M)	– Par manipulation directe d'objets graphiques (cliquage et pointage) *via* des générateurs d'interfaces graphiques	●		● *(Uniquement pour la comparaison maquettage en dynamique de Pacbase et manipulation directe de NSDK)*
Simulation de la maquette	– Affichage alterné de l'écran simulé et de sa grille de définition	– Affichage simultanée de la fenêtre simulée et de l'éditeur graphique (multifenêtrage)	●		
Comportement de la maquette simulée	– Maquette statique	– Maquette dynamique (réactive aux actions de l'utilisateur)	●		
Composants de maquettage	– Rubriques, entités	– Objets graphiques (icônes, boutons, menus), événements,	●		
Mode de conception des interfaces	– Procédural (selon le canevas des grille de saisie)	– Empirique (absence de canevas	●		

Figure 38

Tableau comparatif des caractéristiques ***graphiques*** *des environnements de conception*
(élaboré à partir de l'avis des experts techniques)

a) Dans le cadre de l'interaction homme-machine avec NSDK, il ne s'agit plus de manipuler un dispositif, *via* la saisie d'une chaîne de caractères et d'un langage spécifique, mais de commander le système par l'intermédiaire d'une souris et sur le principe du "*Pointage et Cliquage*". Passer d'un mode à l'autre requiert l'acquisition d'habilités opératoires spécifiques. Par ailleurs, si dans un cas, l'accès à Pacbase est indirect du fait de son langage de commande complexe, dans l'autre cas, l'informaticien exerce un contrôle direct sur l'application par la souris. En

somme, il a la possibilité de mettre directement en application ses buts d'action, sans passer par une phase intermédiaire de codage.

b) D'autres changements touchent les ***supports de maquettage*** : on a vu en effet que Pacbase imposait une grille de saisie et une codification complexe pour construire la maquette (-CE), alors que NSDK proposait un éditeur graphique avec de nouvelles conduites de conception : *sélection*, *positionnement* et *programmation d'objets graphiques*. De plus, ces différents supports induisent un style de maquettage très différent entre eux : rigide et prescrit dans le cas de la grille de Pacbase ; souple et empirique avec l'éditeur graphique de NSDK.
Par ailleurs, Pacbase propose également une technique de maquettage ("*maquettage dynamique*") qui semble, par certains côtés, proche de celle de NSDK. L'informaticien site-central appelle et positionne directement des rubriques sur un écran vierge. Le concepteur NSDK utilise sa souris pour disposer les objets graphiques. Au final, la distance sémantique et opératoire serait donc plus ténue entre le maquettage dynamique et le maquettage graphique qu'entre ce dernier mode et le maquettage par "-CE".

c) Une autre différence notable se trouve au niveau d***es modes de simulation*** de la maquette. Avec Pacbase, la simulation des écrans donne une maquette inerte ; tandis que la simulation sous NSDK affiche une maquette dynamique et réactive aux actions de l'informaticien. Ces possibilités de simulation représentent un véritable atout pour la résolution car l'informaticien peut à tout moment vérifier que le comportement de la maquette correspond bien à son projet de conception, et modifier le cas échéant sa procédure.

d) Enfin, ***le vocabulaire technique*** employé dans le cadre du maquettage change également. L'informaticien doit réapprendre tout un ensemble de codifications anglo-saxonnes associées aux boutons, aux événements, aux fenêtres, aux librairies de NSDK. Cela dit, le fait que ces codes soient accessibles par des menus et des boîtes de dialogues minimisent fortement les efforts de mémorisation.

2.1.5 Les caractéristiques logiques des environnement de conception étudiés

Ces caractéristiques correspondent aux paradigmes de conception "*transactionnels*" *Vs* "*événementiels*" associés respectivement à l'AGL procédural du site-central et au L4G orienté objet du client serveur. Chaque paradigme induit une certaine manière de structurer et de façonner l'interface d'une application.

2.1.5.1 Un paradigme de conception transactionnel et synchrone sur le site-central

Le paradigme transactionnel se réfère à la synchronisation des tâches et des interactions homme/machine Ainsi, une application produite selon cette logique fonctionne selon un mode conversationnel particulier :

"*Requête du système –> réponse de l'utilisateur –>*
traitement –> affichage de la réponse"

L'application produite avec Pacbase suit ce type de logique prescriptive. Elle détermine l'ordonnancement des opérations à réaliser et définit l'introduction des données (à la base d'une transaction). L'informaticien introduit un déterminisme artificiel, en termes d'enchaînements d'écrans, qui structure l'interaction homme/machine et restreint les choix de l'utilisateur final.

2.1.5.2 Un paradigme de conception événementiel et asynchrone sur client serveur

Ici, l'ordonnancement des opérations n'est plus déterminé *a priori* lors de la conception de la maquette, mais dépend essentiellement de l'arrivée des données et des actions de l'utilisateur final. Face au caractère imprévisible de l'interaction, l'informaticien est alors obligé d'établir un dispositif capable d'accueillir les requêtes lorsqu'elles arrivent, de les relier au(x) traitement(s) concerné(s) (affichage d'une fenêtre et fermeture d'une autre), et de rétablir toutes les informations utiles dans l'état où elles étaient lors de l'émission du message précédent. En somme, l'application doit savoir s'adapter à la conduite de l'utilisateur final. Cette logique événementielle implique de nouvelles modalités de conception par rapport à la logique transactionnelle du site-central. Il s'agit :

a) du choix des boutons graphiques, des fenêtres et de leurs événements associés ;

b) de la programmation cohérente, contextuelle et prévisionnelle des événements ;

c) de l'édition de liens (c'est-à-dire d'une mise en relation des différents modules de l'interface) ;

d) du regroupement du code dans une librairie unique.

Nous allons rapidement revenir sur ces caractéristiques.

a) ***Le choix des boutons, des fenêtres et des événements associés***

Pour composer sa maquette, l'informaticien dispose d'un nombre très important d'objets graphiques (14 contrôles et 7 types de fenêtres différents) parmi lesquels il doit opérer des choix compatibles avec son projet de conception. Il lui faut aussi sélectionner les événements (sur la soixantaine disponible) à associer à ces objets. Un événement correspond à un modèle de comportement exécuté par l'objet graphique choisi :

> *Par exemple :*
> La sélection du bouton "*Fermer*" appellera l'événement "*Terminate*", lequel provoquera la fermeture de la fenêtre ; tandis que l'événement "*Get-focus*" conduira à la prédominance d'une fenêtre sur une autre au cours du multifenêtrage.

L'informaticien doit également programmer ces événements pour déterminer des comportements spécifiques au contexte de l'interaction :

> *Par exemple :*
> Il doit définir très précisément les fenêtres qui s'ouvrent et se ferment suite à la sélection d'un bouton.

Il est très rare qu'une commande n'engendre qu'un seul événement sur la maquette. Plusieurs événements sont généralement associés à ce contrôle ; ce qui pose les problèmes plus spécifiques de gestion contextuelle, prévisionnelle et cohérente de ces événements.

b) ***la programmation cohérente, prévisionnelle et contextuelle des événements***

- *La programmation cohérente des événements :* une particularité de la conception événementielle est le nombre important d'événements susceptible d'être rattaché à un objet graphique. La sélection d'un bouton peut provoquer simultanément l'ouverture d'une fenêtre, la fermeture d'une autre et le début d'un traitement. A ces trois actions correspondent trois événements distincts. Pour éviter les

défaillances de l'interface (un blocage par exemple), l'informaticien doit s'assurer qu'il n'y a pas d'incompatbilité entre ces différents événements.

– *La programmation prévisionnelle des événements :* une autre caractéristique de la conception graphique est la programmation prévisionnelle des événements. En effet, comme nous l'avons souvent dit, une application événementielle se caractérise par le fait que l'utilisateur final impose son propre mode de fonctionnement au système, et non l'inverse. Dans ces conditions, l'informaticien doit envisager tous les actions possibles de cet utilisateur afin d'établir des scénarii d'interaction probables et de programmer les événements en conséquence (Cf. Figure 39).

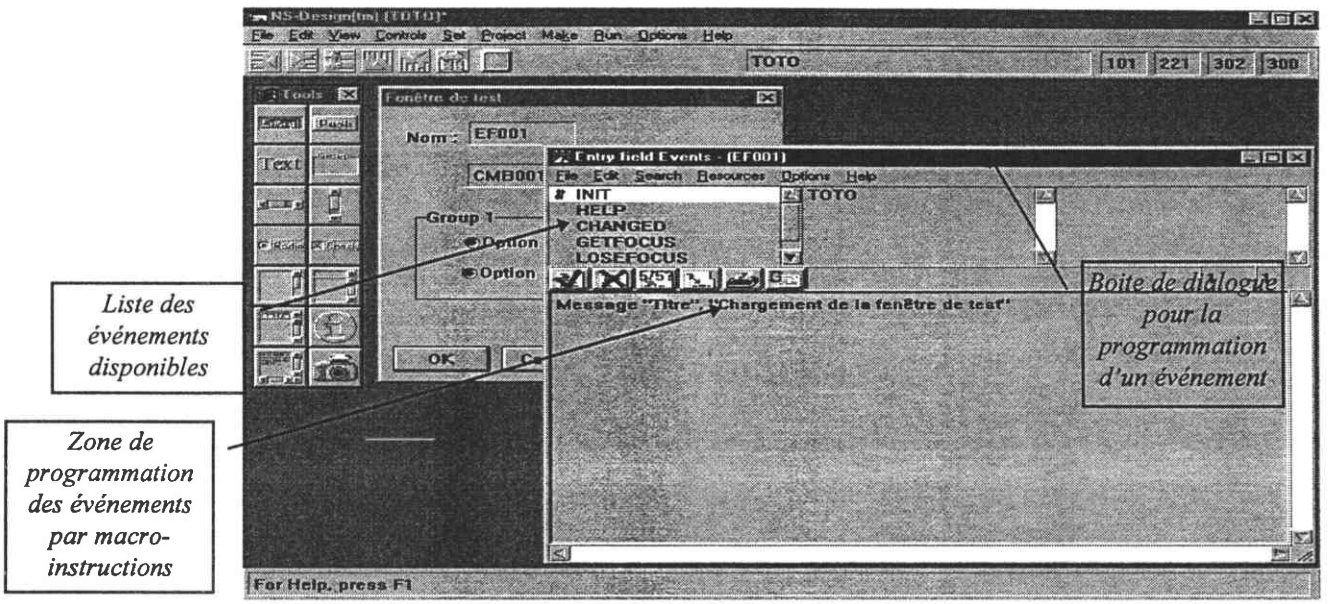


Figure 39
Ecran de maquettage :
boîte de dialogue utilisée pour la programmation d'un événement

– *La gestion des effets contextuels des événements* : cela revient à estimer les conséquences de l'exécution des événements sur les différents composants de la maquette. Ainsi, certains événements n'ont d'effets que dans le cadre restreint de la fenêtre où ils sont déclenchés (effet local), alors que d'autres peuvent avoir des retentissements plus larges (effet global), notamment sur des composants qui ne sont pas affichés ou qui n'ont pas encore été élaborés (comme des fenêtres de dialogue).

Par exemple : La sélection de l'option "Enregistrer" va déclencher simultanément l'événement de sauvegarde, la mise à jour de certaines données dans une bases de données et l'ouverture d'une fenêtre indiquant le succès ou l'échec de la sauvegarde. L'informaticien devra prévoir ces différentes actions et les incorporer à la programmation des événements.

c) L'édition de liens (phase de link)

A l'opposé de la conception monolithique de Pacbase où la totalité de la maquette était élaborée par un seul et même informaticien, la composition de l'interface client serveur est de type modulaire : sa conception est répartie entre plusieurs développeurs : un premier élabore une série de fenêtres, un deuxième s'occupe d'une autre série et ainsi de suite.... Au final, une procédure appelée "*édition de liens*" sera nécessaire pour réunir tous ces composants et former la maquette finale.

d) ***le regroupement du code dispersé dans une librairie unique***

Dernière caractéristique de la conception NSDK, c'est le regroupement du code dans un programme unique. En effet, derrière chaque objet graphique (menu, bouton, icône, fenêtre...) se cache ou un plusieurs événements dont il faut spécifier le comportement par un programme particulier. Plus il existe d'événements, plus les programmes sont nombreux et éclatés dans différents fichiers. La méthode est alors de réunir tout ce code dans un fichier unique pour simplifier la maintenance. Par comparaison, Pacbase dispose d'une fonction automatisée qui prend en charge cette tâche.

2.1.5.3 Synthèse comparative

Ce tableau présente la synthèse des caractéristiques logiques de chaque environnement de conception utilisé.

CARACTÉRISTIQUES LOGIQUES	ENVIRONNEMENT SITE-CENTRAL	ENVIRONNEMENT CLIENT-SERVEUR	INDICE DE NOUVEAUTÉ	INDICE D'ANALOGIE	INDICE DE SIMILARITÉ
Paradigme de conception	– Transactionnel	– Evénementiel	●		
Modèle de l'utilisateur final	– Ignoré (application impose sa logique à l'utilisateur)	– Consacré (application construite selon la logique "supposée" de l'utilisateur)	●		
Programmation cohérente des comportements de la maquette	– Non Pertinente	– Prépondérante (*via* les événements)	●		
Programmation contextuelle des comportements de la maquette	– Gestion seulement des effets locaux du code	– Gestion des effets locaux et contextuels des événements	● (Pour les effets globaux)	● (Pour les effets locaux)	
Programmation prévisionnelle des comportements la maquette	– Non Pertinente	– Prépondérante	●		
Dispersion du code	– code rassemblé dans un programme associé à un écran : pris en charge automatiquement par Pacbase	– code dispersé dans plusieurs fichiers : intervention manuelle de l'informaticien pour rassembler ce code dans un programme unique	●		
Mode de conception de la maquette	– Monolithique	– Modulaire : édition de liens nécessaire par intervention humaine	●		

Figure 40
Tableau comparatif des caractéristiques ***logiques*** *des environnements de conception*
(élaboré à partir de l'avis des experts techniques)

a) ***Le passage du paradigme de conception transactionnel vers le paradigme événementiel*** constitue sans nul doute la rupture la plus manifeste de cette migration informatique. Le concepteur doit dorénavant élaborer des interfaces qui s'adaptent aux actions de l'utilisateur final, et non plus qui les orientent. Ce qui laisse supposer que l'informaticien devra se livrer à une profonde transformation de ses structures de pensées pour aborder de manière résolument différente cette nouvelle tâche de conception.

a) De même, ***les modalités de programmation de l'interface client serveur (cohérente, prévisionnelle et contextuelle)*** sont tout à fait inédites par rapport à celles plus traditionnelle du site-central. Aussi, l'informaticien sera-t-il sans doute obligé d'acquérir de nouveaux modes de raisonnement pour maîtriser ces principes du maquettage. Seule la gestion locale des événements apparaît être une procédure commune aux deux environnements.

b) On note également que l'avènement de la logique événementielle conduit à centrer la conception sur l'utilisateur final. En site-central, ce dernier était totalement ignoré. Dès lors, l'informaticien doit mettre en place de nouvelles règles de coopération pour dialoguer avec cet utilisateur.

c) Tout aussi important est le ***mode de composition de la maquette*** qui conduit soit à une construction "*monolithique*" dans le cas des applications site-central, soit à une composition "*modulaire*" dans le cas des applications client serveur. La phase de liens qu'impose cette dernière technique est une tâche informatique originale qui n'a pas d'équivalent sur Pacbase. Aussi, de nouveaux savoir-faire sont requis pour contrôler cette procédure.

d) Enfin, dernière évolution remarquable ; celle du ***regroupement du code.*** Pacbase centralise automatiquement le code dans un fichier unique alors que NSDK laisse cette tâche à l'informaticien. Cette procédure demande également des connaissances spécifiques.

2.1.6 Synthèse - Discussion

Malgré l'originalité de l'environnement client serveur, on a vu que celui-ci présentait certaines similarités logiques et fonctionnelles avec l'environnement site-central. Nous pensons que ces "*convergences techniques*" peuvent jouer le rôle de passerelles conceptuelles, en permettant au débutant de comprendre le nouvel environnement (client serveur) à partir de connaissances qu'il détient déjà de son ancien système de conception (site-central). En termes plus clairs, ce développeur aurait la possibilité de réutiliser d'anciens apprentissages en prenant soin, toutefois, de les ajuster aux caractéristiques du nouveau contexte technique. Ces similarités seraient en quelque sorte le déclencheur des transferts d'apprentissage positifs. Dans ce cas, on serait en présence d'un processus de **réapprentissage.**

De même, à supposer que l'informaticien ait identifié des analogies entre certaines caractéristiques des deux environnements, un transfert analogique d'anciennes structures de connaissances pourrait très bien être initié.

Cependant, l'avènement du client serveur induit plus souvent des innovations technologiques et fonctionnelles, qui le démarquent fondamentalement du site-central. Le transfert d'anciennes connaissances doit dès lors être abandonné au profit de la construction de nouveaux modes d'action et d'interaction. Dans ces conditions, il s'agit **d'apprendre** et non plus de réapprendre.

Le tableau ci-dessous rappelle les différents niveaux de convergence/divergence observés entre les deux systèmes, et envisage divers types d'apprentissage qui peuvent être requis pour maîtriser les nouvelles caractéristiques de l'environnement cible (Cf. Figure 41).

CARACTÉRISTIQUES TECHNIQUES CONDUITES D'APPRENTISSAGE ENVISAGÉES	NOUVEAUTÉS TECHNIQUES *RUPTURE TECHNOLOGIQUE ENTRE LES DEUX ENVIRONNEMENTS → APPRENTISSAGE NÉCESSAIRE*	ANALOGIES TECHNIQUES *CONTINUITÉ TECHNOLOGIQUE → RÉAPPRENTISSAGE PAR UTILISATION DIRECTE DE L'EXPÉRIENCE ACQUISE*	SIMILARITÉS TECHNIQUES *PERSISTANCE DE CERTAINS TRAITS TECHNIQUES → RÉPPRENTISSAGE PAR AJUSTEMENTS DE L'EXPÉRIENCE PASSÉE*
CARACTÉRISTIQUES FONCTIONNELLES			
– Les instructions de programmation			les commandes et les macros instructions existent toujours, mais sous d'autres formes
– Paradigme du code généré par les outils		Procédural (Cobol et C)	
– Les dispositifs de stockage des entités de programmation			Librairies (NSDK) ou bibliothèque (Pacbase)
– Propriété du langage utilisé pour la conception	Langage interprété (NCL) *Vs* langage compilé (Pacbase)		
– Canevas de programmation	Squelette de programmation *Vs* Editeur graphique		
CARACTÉRISTIQUES GRAPHIQUES			
– Techniques de maquettage	Maquettage par grille de saisie (-"Ce") de Pacbase *Vs* Manipulation directe avec NSDK		Maquettage dynamique de Pacbase et par positionnement graphique avec NSDK
– Nature de l'interface de communication	Alphanumérique de Pacbase *Vs* Graphique avec NSDK		
– Mode de manipulation de l'interface	Codification *Vs* Manipulation directe		
– Entités de programmation manipulées	Rubriques et codes *Vs* objets graphiques		
CARACTÉRISTIQUES LOGIQUES			
– Programmation des commandes et des événements	Gestion locale des effets des commandes (Pacbase) *Vs* gestion contextuelle, prévisionnelle et cohérente des effets des événements (NSDK)	Gestion locale des effets des commandes(sur Pacbase et sur NSDK)	
– Le regroupement du code	Rassemblement du code automatisé (pris en charge par Pacbase) Vs regroupement manuel (effectué par l'informaticien sur NSDK)		
– Le paradigme de conception	Transactionnel (Pacbase) *Vs* Evénementiel (NSDK)		
– Le mode de conception des application	Monolithique (Pacbase) *Vs* modulaire (NSDK)		
CARACTÉRISTIQUES ARCHITECTURALES			
– Configuration de l'architecture	Centralisée *Vs* Distribuée		
– Place de l'architecture dans la conception	Utilisateur Ignoré *Vs* Consacré		
– Fiabilité de l'architecture	Robuste *Vs* Instable		
– bases de données			DB2 et DL1 sur site-central et Sybase en client serveur : même paradigme fonctionnel

Figure 41

Tableau comparatif des caractéristiques techniques des environnements source et cible

En somme, selon les niveaux de proximité technique des environnements, les apprentissages des informaticiens pourraient prendre différentes expressions :

1. d'abord, lorsqu'il existe une *analogie* entre les caractéristiques techniques des deux environnements, les transferts d'apprentissage ont toutes les chances d'être positifs grâce à un raisonnement de type analogique qui, sur la base des aspects désignés comme identiques aux deux contextes, déclenchera la réutilisation des connaissances appropriées. Par exemple, la compréhension de "cobol" et de "C" ne nécessitera pas de nouveaux apprentissages puisque tous deux appartiennent à la catégorie des paradigmes procéduraux.

2. Lorsqu'il y a *persistance technique* (similarité) entre certaines caractéristiques des environnements, on peut s'attendre à ce que l'informaticien se réfère spontanément à ses anciennes connaissances. Mais sans un processus d'ajustement circonstancié de ses connaissances, l'informaticien risque fort de rencontrer des difficultés pour maîtriser les nouveaux aspects techniques.

3. Enfin, lorsque la *rupture technologique* est effective, il convient pour l'informaticien de procéder au renouvellement de ses connaissances par des apprentissages originaux. Leur réutilisation conduira *de facto* à des transferts négatifs. Par exemple, pour élaborer des interfaces en multifenêtrage, l'informaticien doit abandonner sa logique de conception transactionnelle pour adopter une démarche événementielle. Sans quoi, les maquettes développées ont les plus grandes chances d'être incompatibles avec les règles de conception client serveur.

2.2 L'ORGANISATION DE L'ACTIVITÉ DE MAQUETTAGE

L'objectif de cette deuxième partie est d'exposer le déroulement de l'activité de maquettage adoptée par les informaticiens dans chaque contexte informatique : d'abord en environnement site-central, puis client serveur.

2.2.1 Méthode employée

a) Méthode de recueil de données

La méthode a consisté, tout d'abord, à prendre connaissance des caractéristiques générales de l'activité de maquettage par le biais ***d'entretiens exploratoires***. Ces interviews, d'une durée moyenne d'une heure, ont été réalisées auprès de 9 informaticiens experts[40] en Pacbase et de 7 expert en NSDK. Nous leur demandions, entre autres choses[41], de décrire précisément le déroulement de l'activité de maquettage et de préciser les conditions de réalisation des ces opérations de maquettage. On a cherché aussi à identifier les données et les informations dont ils estimaient avoir besoin pour effectuer leur travail.

Cependant, ces entretiens fournissaient une certaine représentation que les informaticiens se font de la tâche à réaliser, et non la réalité. C'est pourquoi, nous avons étayé ces entretiens par des observations effectuées en situation réelle de maquettage. Elles duraient en moyenne 60 minutes et ont porté sur 5 informaticiens experts Pacbase et 5 experts NSDK. Les séquences d'observation correspondent à celles que nous avons déjà établies dans la méthode d'intervention (Cf. partie 1.2.3. du chapitre 3).

Parmi le nombre important de données[42] qui ont été récoltées, nous avons retenu les informations relatives à l'organisation de l'activité, à la nature et au déroulement des séquences opératoires, aux types d'informations recherchées, aux relations entre collègues et aux documents utilisés.

---

[40] Les caractéristiques biographiques de ces échantillons sont présentées dans la partie 3.1.2.4

[41] On trouvera la liste exhaustive des thèmes abordés durant ces entretiens en Annexe de cette recherche (Cf. Annexe II).

[42] Les aspects de l'activité qui ont retenu notre attention durant ces observations sont présentés dans l'Annexe III

A la suite de ces observations, des entretiens plus ciblés ont également été réalisés avec les informaticiens sur certaines caractéristiques remarquées de leur activité ; comme par exemple, la manière dont s'effectuait le paramétrage d'une rubrique.

c) Méthode d'analyse des données

La méthode d'analyse de ces données a été la suivante : nous avons tout d'abord reconstitué le déroulement de l'activité de maquettage à partir des observations effectuées sur le terrain. L'idée était de privilégier les caractéristiques qui se retrouvaient fréquemment dans ces observations pour élaborer un modèle moyen de l'activité. Puis, nous avons recherché dans les entretiens réalisés (exploratoires et ciblés) des extraits de corpus qui pouvaient illustrer les faits observés. Nous en présentons d'ailleurs des exemples dans cette partie.

Nous avons aussi cherché à déceler les écarts existant entre la description subjective de la tâche faite par les informaticiens et nos observations sur l'activité réelle.

2.2.2 Organisation de l'activité de maquettage avec Pacbase

Cette étape de maquettage se déroule juste après la phase d'analyse détaillée. Elle marque la transition entre le domaine de l'analyste et celui du programmeur : ce dernier développe les spécifications que l'analyste a définies dans un dossier de conception. Il s'agit de la configuration générale des écrans, de leur enchaînement, des programmes à développer, des données à rechercher.

D'après nos observations, l'activité de maquettage alterne 6 phases successives (Cf.Figure 44) :

1. La première action du concepteur consiste à identifier dans les sources externes du dictionnaire une application dont l'interface se rapproche le plus possible des spécifications du dossier de maquettage. Etant donné la très grande diversité des projets stockés, l'informaticien réussit presque toujours à retrouver au moins une maquette admissible pour la tâche de maquettage.

   Par la suite, il procède à **l'examen** de l'interface récupérée et détermine le type de corrections et/ou d'aménagements qu'il souhaite effectuer. Pour ce faire, il visualise la grille de définition de ces écrans (commande "-CE"). Il a la possibilité de supprimer, de modifier ou d'ajouter des rubriques en les codant dans les zones prévues à cet effet. (Cf. Figure 43).

> *« L'avantage du "-CE" est que l'on peut refaire un écran à partir de quelque chose qui existe. C'est-à-dire que l'on se positionne sur un écran qui nous plaît. On lui donne un nouveau nom. On fait "entrer" et là, on va se baser sur quelque chose qui existe pour construire l'écran ».*

2. La **conception de l'écran** se déroule selon deux étapes complémentaires : D'abord, l'informaticien détermine dans la grille de maquettage la nature des éléments à afficher (principalement des rubriques[43] et des libellés[44]) (Cf. Figure 42) et définit leurs coordonnés de positionnement à l'écran (abscisse et ordonnée). Ensuite, il spécifie les attributs physiques de ces entités (leur couleur, leur format majuscule/minuscule) au moyen de codifications appropriées (Cf. Figure 43).

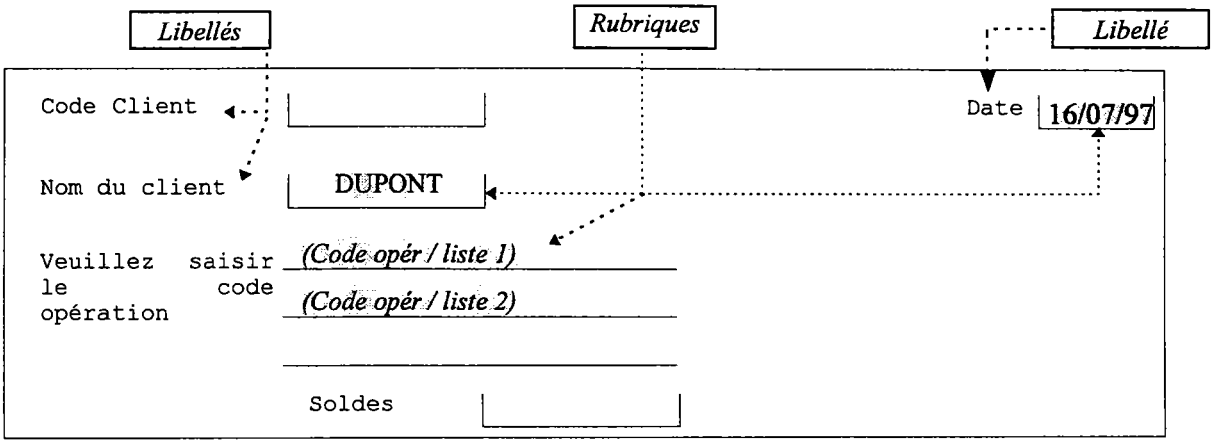


Figure 42
Exemple d'écran récupéré avec des libellés et des rubriques

4. Chaque intervention de l'informaticien sur la grille de maquettage se ponctue par la **visualisation** de l'écran ("-Sim"). L'écran obtenu ne peut être manipulé car c'est une image statique qui s'affiche. Néanmoins, sa représentation permet à l'informaticien de s'assurer que les changements définis sur la grille ne perturbe par l'agencement général de l'écran réutilisé. Rappelons en effet que l'interface alphanumérique de Pacbase interdit le multifenêtrage, et donc la présentation simultanée de la grille de définition et de son écran. L'informaticien doit alors passer alternativement d'une représentation à l'autre. Ces vérifications terminées, il peut se consacrer à l'étape suivante qui consiste à déterminer la fonction des rubriques (Cf. Figure 43).

[43] Rappelons que ka *Rubrique* est la donnée élémentaire d'un écran. Elle peut être consultée, renseignée ou modifiée par l'utilisateur. Elle constitue l'entité fondamentale du maquettage sur Pacbase.
[44] *Libellé* : c'est un commentaire fixe qui précède une zone saisissable par l'utilisateur. Il lui indique la nature de l'information consultée ou à saisir.

« Là par exemple, cette rubrique ne me plaît pas et je veux la modifier. Ce que je fais, c'est que je vais ici avec mon curseur sur la zone de la rubrique à modifier. Je me mets en majuscule, je modifie, je tape « -sim », et là, l'écran de la maquette utilisateur s'affiche et je peux voir si c'est bien modifié. »

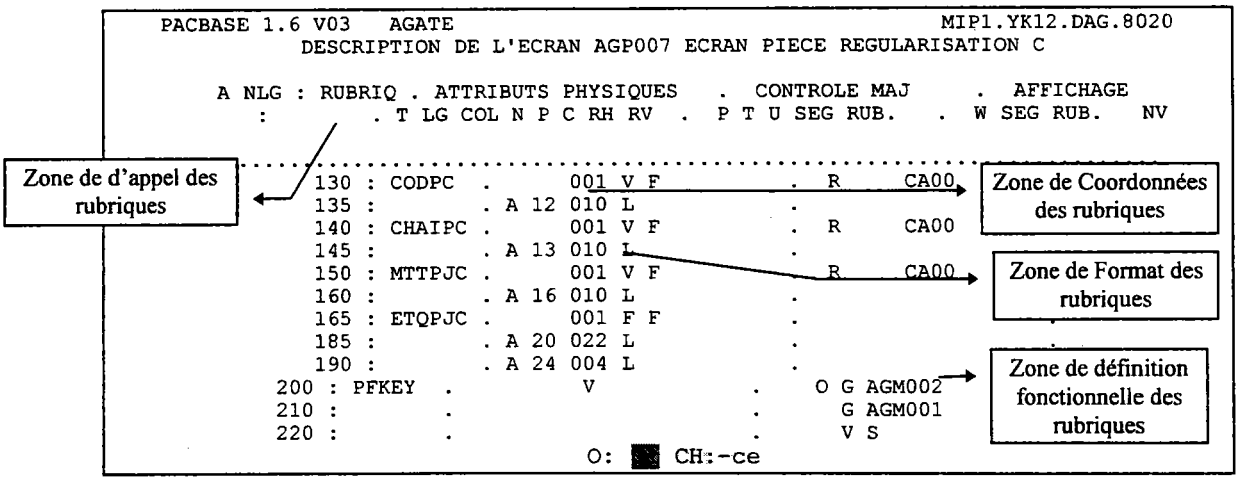


Figure 43
Grille de maquettage "-CE" : codification des écrans

On a remarqué aussi que les informaticiens les plus chevronnés utilisaient une autre technique de maquettage qu'ils qualifiaient de dynamique. Il s'agit en fait de construire l'écran en appelant directement les entités de maquettage sur un écran vierge grâce à une codification particulière (nom de la commande de maquettage : "-M").

4. L'objet de cette quatrième étape est de paramétrer les fonctions des rubriques. L'informaticien précise :
   - si la zone de cette rubrique peut être saisissable ou non par l'utilisateur ;
   - si elle est obligatoire ou non ;
   - quels sont les types de contrôle qui vont s'exercer sur ces rubriques (par exemple : "*vérification de la cohérence de la date*") ;
   - quelles sont les opérations que l'utilisateur pourra réaliser sur cette rubrique
   - etc.

Lorsque le paramétrage de ces rubriques est enfin terminé, l'informaticien débute la réalisation d'un autre écran, et ainsi de suite jusqu'à obtenir l'ensemble de l'interface.

5. Une dernière étape du maquettage consiste à déterminer la **cinématique des écrans**, c'est-à-dire à fixer leur ordre d'enchaînement tel qu'il s'imposera à l'utilisateur final.

6. Le **test final** de la maquette interviendra après la compilation du code. Il sera alors possible "d'animer" la maquette, ou tout au moins de faire défiler les écrans dans l'ordre fixé par l'informaticien.

7. La maquette finale est présentée au client pour **validation**. C'est d'ailleurs la seule fois où l'utilisateur est véritablement consulté pour donner son avis. Si aucune critique n'est émise (cas le plus fréquent), le développeur commence alors la programmation des traitements de l'application et on arrive ainsi au terme du processus de maquettage.

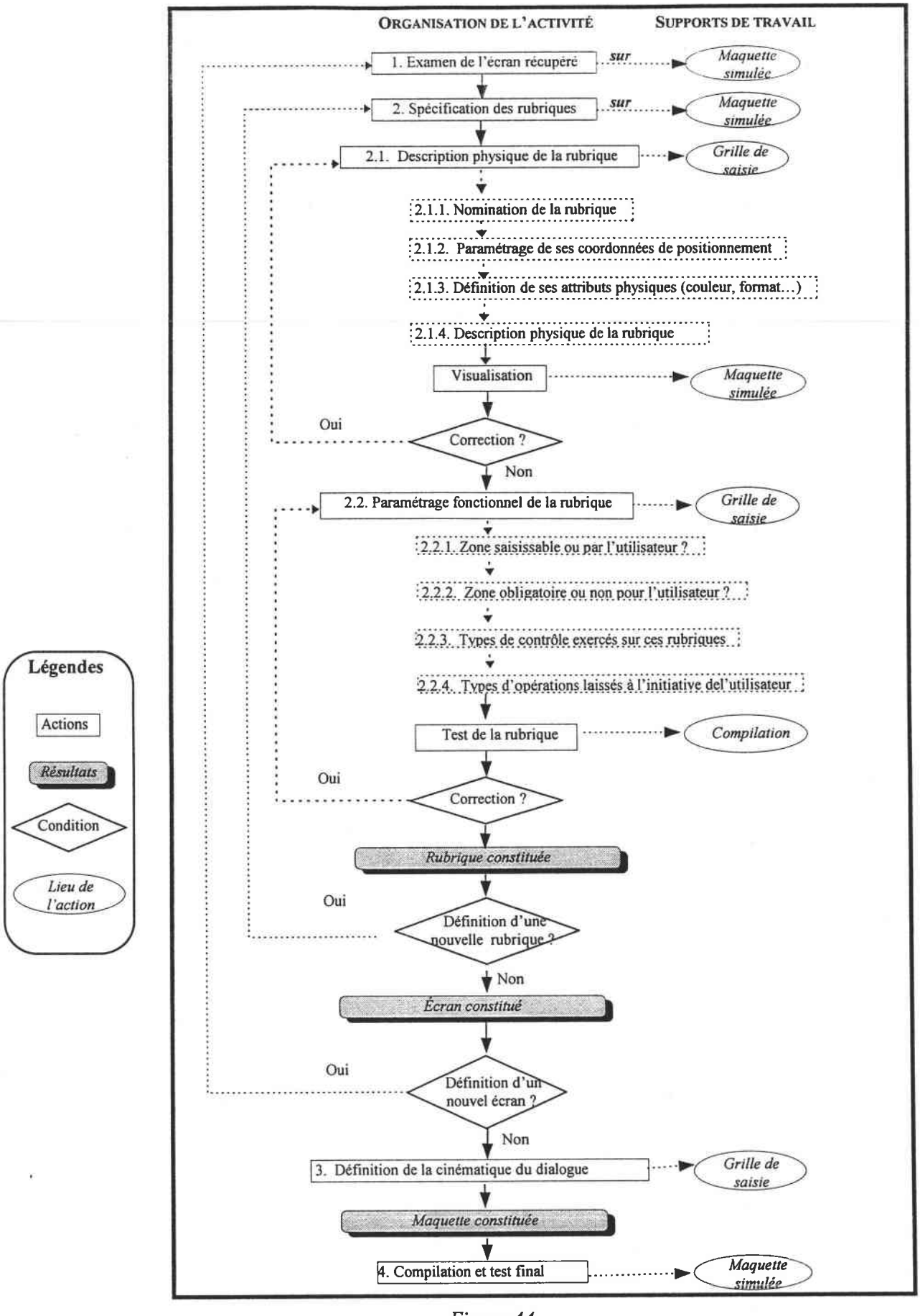


Figure 44
Activité de maquettage sur Pacbase

2.2.3 Organisation de l'activité de maquettage avec NSDK

Plus haut, nous avons dit que le maquettage en environnement site-central se caractérisait par la richesse des projets de conception réutilisables. Or, parce qu'il est encore assez récent, NSDK n'offre pas un tel vivier de ressources techniques. L'informaticien est donc contraint de créer une solution de maquettage *ex-nihilo*. Dans ces conditions, les étapes de conception se succèdent au rythme des étapes suivantes : sur la base des spécifications détaillées, mais aussi, après de nombreux échanges avec l'utilisateur final pour comprendre ses besoins et appréhender le contexte de son travail, l'informaticien est prêt à s'engager dans la tâche de maquettage.

> *« A partir du moment où l'on sait quelle est la population d'utilisateurs, s'ils sont tous sur le même site, s'ils ont le même profil, s'ils vont utiliser ces objets de la même manière ou selon des cycles de vie des données, on peut commencer à réfléchir sur la maquette. »*

1. La réalisation effective de l'interface ne sera pas immédiate. L'informaticien se livre d'abord à une phase de description de chaque fenêtre sur le papier. On notera d'ailleurs que ce sont les fenêtres et non plus les écrans qui deviennent les entités graphiques de référence. Ce document papier fait office de **scénario de maquettage**. Il renferme des informations qui vont guider l'informaticien dans la conception de l'interface, notamment :
   i. une *description générale de l'enchaînement des fenêtres* : quelles sont les liens de dépendance entre les fenêtres (fenêtre mère et ses fenêtres filles) ;
   ii. le *type et le nom de chaque fenêtre utilisée* : cela peut aller d'une simple fenêtre d'information (par exemple, "*Fichier sauvegardé*") à une fenêtre plus riche qui propose des menus, des icônes et des graphiques ;
   iii. *les attributs de ces fenêtres*, c'est-à-dire les objets graphiques qui les composent comme les boutons, les ascenseurs, les icônes, les champs de saisie ;
   iv. *l'état courant de ces attributs ;* c'est-à-dire les libellés qui les qualifient et éventuellement les données qui les alimentent ;
   v. enfin, *le comportement de chacun de ces contrôles*, c'est-à-dire les actions qu'ils vont générer sur la maquette (ouverture/fermeture de la fenêtre, des boîtes de dialogue...).

> *« Sur le papier, je mets les éléments de la fenêtre, et je précise ce que cela doit faire quand on les sélectionne, quand on appuie sur les boutons. Par exemple, quand ils sont exécutés, cela doit griser tels champs. Cela doit faire ça, initialiser ça, puis enlever le relief, remettre la couleur noire... »*

L'étape suivante consiste à réaliser cette fenêtre avec l'éditeur graphique de NSDK :

> *« Donc, à partir du moment où on a tout ça, je dirais que l'on a suffisamment de billes, on peut commencer à faire le maquettage »*

2. et 3. L'informaticien commence par **sélectionner** une fenêtre en lui donnant un nom. Puis, il choisit les objets graphiques dans une boîte à outils et les dispose sur la fenêtre par manipulation directe. C'est l'étape **d'habillage ou de composition** de la maquette.

4. L'informaticien veille également à **agencer** ces objets en respectant les normes de présentation graphique édictées par le service méthode (étape de **structuration de la fenêtre**). On notera d'ailleurs que l'évaluation graphique se fait directement sur l'écran de maquettage, et non par l'intermédiaire d'un écran de simulation comme sur Pacbase.

   > *« Au niveau de la description des écrans, il y a beaucoup plus de travail en graphique qu'en 32-70* [mode textuel]. *J'ai même passé plus de temps rien que pour positionner les champs.(...) Les réglages de présentation sont fastidieux, au pixel près».*

5. Une fois la présentation établie, l'informaticien se consacre la **"programmation fonctionnelle"** de ces objets. Il s'agit en fait de caractériser le comportement spécifique de chaque objet, selon trois étapes :

   i) L'informaticien attribue à chaque objet un ou plusieurs événements qu'il sélectionne dans une liste.

   > *« Sur ce bouton "OK", on va associer un événement "exécuter". donc, dans la librairie de la fenêtre, on va dire en "OK" = "executed". »*

   i) Puis, il détermine le comportement que vont prendre ces événements en programmant leurs fonctions ou en réutilisant des fonctions préfabriquées disponibles sur des librairies spécialisées :

   > *« On doit tout coder, c'est-à-dire que l'on va associer au bouton "OK" l'événement "Executed", et derrière programmer le code permettant soit un traitement, soit une ouverture d'une autre fenêtre soit le grisage de certaines options du menu, ou toutes ces choses en même temps. »*

i) L'informaticien précise enfin un certain nombre de caractères propres à ce contrôle :

« Derrière chaque objet, on donne des informations diverses : on donne son nom qui va servir en programmation, on a aussi des descriptions du champ, le type de caractère que peut recevoir ce champ, ainsi de suite... ».

6. Le développeur procède au **test** de cette fenêtre grâce à la fonction de simulation directe. La fenêtre devient alors active et peut être visualisée à côté de l'éditeur : il est possible de jouer avec ses attributs pour naviguer dans l'interface. On peut ainsi fermer ou appeler une fenêtre, faire fonctionner les ascenseurs, sélectionner les boutons. L'évaluation portera donc à la fois sur les caractéristiques externes (aspects de surface) et internes (comportements et réactions) de l'interface. De cette manière, l'informaticien s'assure que les événements programmés engendrent bien les comportements attendus. En effet, le manque d'expérience du novice le conduit bien souvent à faire des choix qui relèvent plus souvent du tâtonnement que d'une réelle stratégie éprouvée de conception. Le recours régulier à la simulation représente dès lors un moyen efficace pour évaluer et affiner le maquettage :

« On peut tester les boutons en temps réel. On n'est pas obligé de tout compiler, on peut le faire au fur et à mesure. On essaie en fait. C'est du tâtonnement, c'est de l'ajustement un petit peu à chaque fois (...) Au début, on essaie quelque chose, on fait tourner la maquette. On regarde si on a bien obtenu ce que l'on voulait obtenir. »

7. L'une des dernières étapes du maquettage consiste à **rassembler** tous les fichiers de codes des différents événements dans un programme unique. Cette opération va simplifier la maintenance de la maquette.

« On va regrouper le code dans une seule librairie NCL, plutôt que d'avoir plusieurs parties de codes dispersées dans la fenêtre. On fera juste des appels de fonctions pour exécuter le bouton "OK". »

Dans le cadre de ce projet, l'informaticien est également amené à réfléchir sur l'organisation qu'il doit donner à l'architecture client serveur. Ainsi, selon les performances recherchées, la taille de la maquette, le nombre d'utilisateurs finaux concernés..., faut-il mieux attribuer la gestion de l'interface au poste client ou au poste serveur ?

« Quand on arrive aux spécifications techniques, on se dit : tout ça, c'est bien beau ; on a fait le maquettage et on voit comment cela marche. Maintenant, cette application là, il faut savoir comment techniquement on va passer les données, où est-ce qu'on va les mettre, sur le client, sur le serveur ? »

8. L'activité de maquettage se terminera par une double **validation** de l'interface :
   - d'abord, par le service méthode qui se prononcera sur la qualité ergonomique et fonctionnelle du produit,
   - puis, par les utilisateurs finaux qui donneront leur avis sur la convivialité et l'utilisabilité de cette interface.

« Le maquettage va nous aider aussi pour redéfinir ou affiner les règles de gestion, parce qu'en voyant la maquette et les écrans qui s'enchaînent, l'utilisateur va dire, moi, je veux telles données là. »

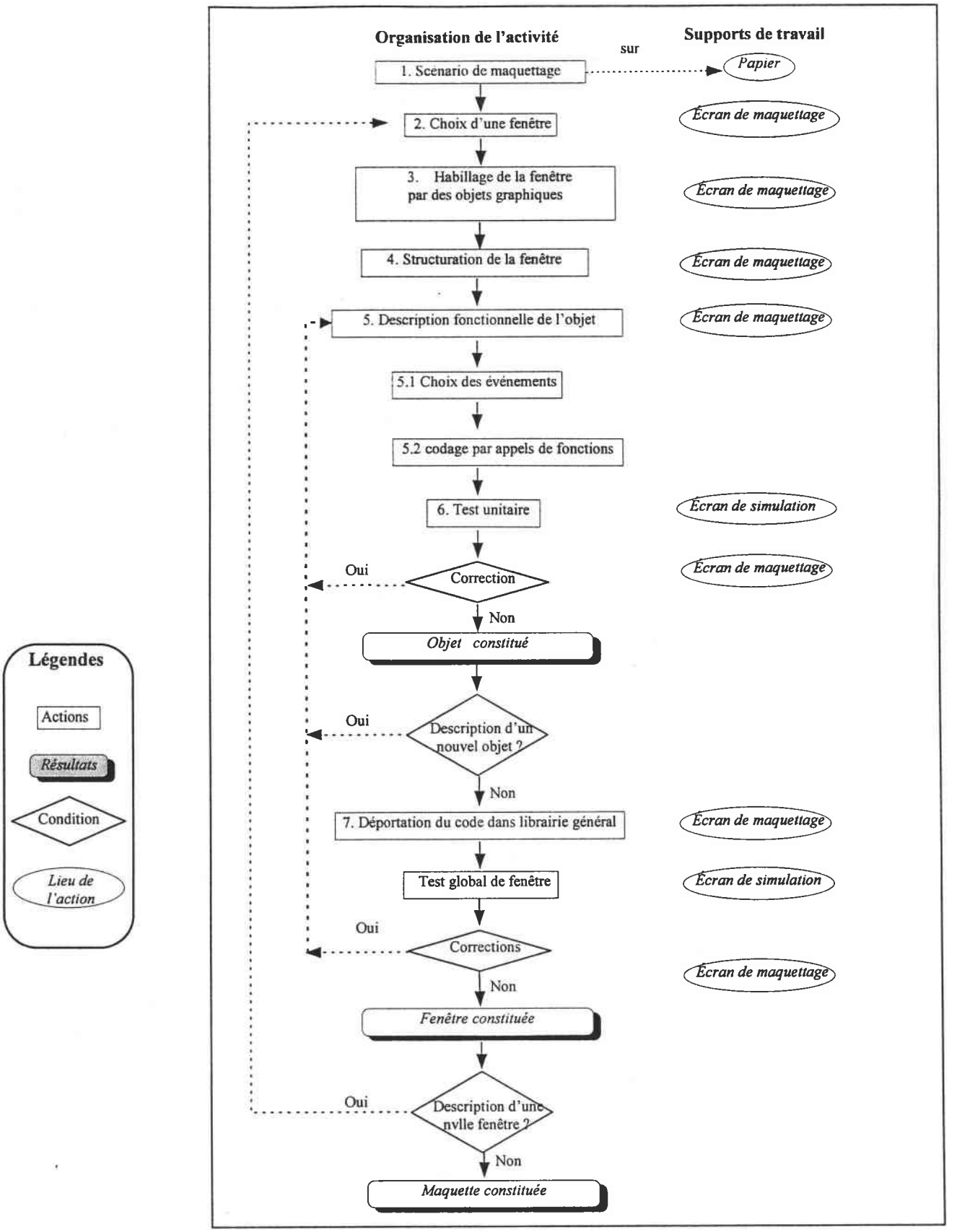


Figure 45
Organisation de l'activité de maquettage en client serveur

2.2.4 Synthèse - Discussion

La comparaison de l'organisation des activités de maquettage nous conduit à relever les points suivants :

1. *De nouvelles activités ont fait leur apparition.*

   En effet, la tâche de maquettage avec NSDK ne consiste plus seulement à composer graphiquement des fenêtres ; elle se double d'une tâche de programmation du comportement de l'interface qui n'existait pas dans Pacbase.

   Avant de poursuivre, il est utile d'indiquer que le nouvel environnement de conception n'a pas seulement généré de nouvelle tâches de conception, mais qu'il a également contribué à une véritable revalorisation du maquettage. En effet, dans le site central, seules les étapes d'analyse, situées en amont de la conduite de projet, étaient réellement reconnues par les informaticiens. Deux raisons expliquaient cet engouement : d'une part, les contacts réguliers et directs avec le client procuraient aux analystes une légitimité au sein de l'équipe projet. D'autre part, en déterminant les spécifications à développer, l'analyste pouvait influencer et contrôler l'activité des programmeurs. Ce dernier s'apparentait alors à un simple exécutant sans réel pouvoir, ni responsabilité dans la conception. D'où cette déconsidération des étapes de réalisation (maquettage et programmation) :

   > *« C'est plus le côté analyse, le côté recherche qui nous intéresse, plutôt que de pondre de la ligne, de coder. Cela ne fait plaisir à personne. Comme pour une secrétaire, la sténo c'est pfff... ! ! »*

   Cette dépréciation de la réalisation était accentuée par les canevas de conception imposés par Pacbase (squelettes de programmation et grille de maquettage) qui encadraient fermement l'activité du programmeur et limitaient du coup toutes initiatives de développement :

   > *« Pacbase nous donne des choix de menus et on ne peut pas faire autrement que de faire ces choix et d'aller dans ce qui est défini »*

   L'arrivée de l'environnement client serveur a donc eu pour effet de redonner une certaine légitimité à ces étapes grâce :

   - à la simplicité et à la convivialité du système : utilisation ludique *via* l'interface graphique et la souris ;

- l'absence de canevas de conception : pour la première fois de sa carrière, l'informaticien a la possibilité de laisser libre cours à son imagination en créant ses fenêtres et en organisant lui-même son activité, sans l'aide du système ;
- un cycle de vie par prototypage qui a permis à l'informaticien de concevoir dynamiquement l'interface avec l'utilisateur final. Ces contacts réguliers ont contribué à faire reconnaître son activité.

  « Sur site central, les maquettes étaient spécifiées durant l'étude détaillée, avec les spécs [les spécifications]. *En client-serveur, c'est beaucoup moins vrai. Ce sont les échanges réguliers avec l'utilisateur qui permettront de construire et d'affiner la maquette »*

2. *Une activité de conception moins linéaire*

La démarche de conception est beaucoup moins structurée et figée en client serveur qu'en site-central. Des retours en arrière sont désormais possibles lorsque l'utilisateur ou des contingences techniques l'exigent. La souplesse de l'environnement de conception permet ainsi de développer, de tester et de corriger de manière très dynamique chaque projet de maquettage.

« Cette notion de maquettage se fait en étapes cycliques: je dirais par expérience que trois tours de boucle restent relativement suffisants. Mais, il ne faut pas que cela parte dans une spirale infinie (...) ».

3. *Recours important à la simulation*

La simulation de la maquette apparaît être une étape-clef de la conception client serveur à laquelle les informaticiens font régulièrement appel pour progresser dans le maquettage. Ils peuvent à tout moment évaluer la fiabilité d'une fenêtre en testant son comportement par des jeux d'essai. Ils ont aussi la possibilité d'explorer de nouvelles voies de conception en simulant leurs idées de maquettage. En site-central, les phase de simulation sont limitées en raison de leur coût élevé (CPU).

« Dans PAC, il y a avait une grosse phase de maquettage et puis une phase de test. Alors que là, c'est différent : c'est au fur et à mesure que l'on teste. »

4. *Une liberté paradoxalement difficile à gérer* :

On a observé que la liberté de conception recouvrée grâce à l'absence des canevas dans NSDK pouvait s'avérer paradoxalement problématique. En effet, l'éditeur graphique met à disposition, par le biais de différentes boîtes à outils, une collection de techniques d'interaction (objets graphiques, événements, fonctions...) sous forme de composants réutilisables. La conception de l'interface graphique consiste, comme on l'a vu, à rappeler et organiser ces techniques d'interaction. Toutefois, ces boîtes à outils, qui incluent plusieurs centaines de combinaisons possibles, ne fournissent pas de schéma général d'assemblage. Le programmeur reste ainsi confronté à l'utilisation de multiples procédures qu'il doit connaître. Ce sont alors les manuels d'utilisation, et surtout les normes de présentation définies par les cellules méthodes (Dabfi-0) qui vont définir les conventions d'application.

> *« C'est pire que Pacbase dans le sens où il y a plus de choses, plus d'objets, plus de possibilités (...) C'est assez déconcertant car on peut trouver36 000 solutions graphiques. Il y a énormément de choix à faire et d'options à prendre »*

5. *La disparité des offres applicatives réutilisables* :

Une différence significative entre Pacbase et NSDK provient de la quantité de solutions de maquettage réutilisables dans chaque environnement : si en site-central, le programmeur a la possibilité de réutiliser très facilement un ancien projet parmi l'importante réserve accumulée au fil des années dans le dictionnaire; les librairies de NSDK, en revanche, offre très peu de ressources exploitables. L'informaticien est alors contraint de créer une solution de toute pièce. Ce qui nous incite à dire que :

- dans l'environnement site-central, le maquettage consiste à **compléter les trous du squelette** et à redéfinir les partie inadaptées au cas particulier du projet de conception. L'activité de développement est alors guidée par les outils et non par le dialogue, encore moins par la tâche.

  > *« Bon ce qui est courant aussi* [sur Pacbase, NDR[45]], *c'est de reprendre les dossiers existants pour reprendre les idées. Parce que c'est pareil : entre un canevas qui est proposé et puis la réalisation, on a toujours besoin d'exemples. Il est fréquent de garder les dossiers et puis on se dit là dedans, c'est super, c'est bien fait, c'est bien expliqué. Je vais prendre cette partie là.»*

[45] Note du Rédacteur

- en environnement client serveur, l'absence de projets amène l'informaticien à **construire** d'abord des scénarii sur papier avant de débuter la conception graphique. C'est donc une tâche plus laborieuse et moins systématique que précédemment, où les tâtonnements de conception et les simulations se succéderont sur la base d'un processus itératif.

Aussi, selon le contexte technique de conception dans lequel il se trouve, l'informaticien est confronté à deux modes de gestion de son activité :

- un mode **procédural** sur Pacbase : l'informaticien est encadré par le logiciel. Il est astreint à suivre une procédure ;
- un mode **empirique** sur client serveur : l'informaticien définit une démarche, en utilisant une boîte à outils mise à sa disposition.

5. *Elargissement du cercle des intervenants dans le maquettage :*

Nous avons constaté que les informaticiens avaient vu leur rôle et leur fonction se modifier au cours de la migration technologique. Leur pouvoir s'est considérablement réduit au profit d'utilisateurs plus exigeants et de nouveaux acteurs plus expérimentés sur des domaines techniques innovants : prestataires spécialisés en conception client serveur, ergonomes, administrateurs de bases de données, spécialistes réseaux. En fait, si d'un côté l'informaticien fait preuve d'une plus grande autonomie par rapport au système technique (absence des canevas prescriptifs), de l'autre, il dépend de plus en plus ses différents partenaires de conception (interdépendance fonctionnelle et cognitive).

> *« L'utilisateur demande énormément de choses. Il a l'impression qu'il pourra faire ce qu'il veut avec l'application. (...) Avec Pac, c'était pas pareil ; on connaissait bien l'outil donc on pouvait lui répondre. Alors que là avec NSDK, c'est plus difficile pour lui dire non, on n'a pas toujours les arguments. Il veut par exemple pouvoir ouvrir toutes ses fenêtres en même temps. Ce qui ne pouvait jamais nous être demandé avec Pac. »*

Notons enfin que si le pouvoir des informaticiens a diminué au profit des utilisateurs et des autres intervenants, leur champ d'intervention a au contraire augmenté : plus polyvalents, ils interviennent à tous les niveaux du processus de conception, et ne sont plus seulement cantonnés aux tâches d'analyse ou de programmation, comme dans Pacbase.

En définitive, il ressort que les nouvelles techniques de conception client serveur ont conduit à la redéfinition de l'espace d'action des informaticiens, en modifiant notamment :

- *les modalités d'organisation de l'activité* : gestion plus empirique que linéaire et procédurale ;
- *les modalités de conception des solutions de maquettage* : création de solutions originales, et non plus adaptation de solutions récupérées ;
- *les modalités de la coopération* : échanges plus nombreux avec les partenaires de la conception, mais remise en cause de son pouvoir au sein du projet ;
- *les modalités de l'intervention :* réalisation d'une nouvelle tâche de programmation de la maquette et élargissement de son activité à l'ensemble du processus de conception.

Jusqu'à présent, nous avons décrit le contexte technico-organisationnel de notre intervention. Il reste à spécifier les contraintes que ces dispositifs techniques peuvent générer sur l'informaticien.

2.3 L'ANALYSE DES CONTRAINTES ET DES ASTREINTES

Comme nous venons de le constater, l'arrivée de l'environnement client serveur conduit à l'évolution des techniques de conception et provoque la réorganisation de l'activité de maquettage. Ces nouvelles caractéristiques représentent autant de contraintes potentielles qui peuvent s'exercer sur l'informaticien débutant en NSDK.
Mais la gestion de ces contraintes induit aussi un coût mental – en termes de stress, d'épuisement mental, de fatigue ou de charge mentale– que l'on désigne plus généralement par l'expression d'astreinte. Celle-ci est susceptible d'affecter non seulement le déroulement de la tâche (au niveau de la productivité, de la fiabilité du processus, de la qualité des maquettes conçues, de la coopération entre les partenaires), mais également les attitudes de l'informaticien vis-à-vis des nouvelles technologies (rejet des nouveaux dispositifs, interaction homme-machine difficile, résistance au changement). C'est pourquoi, nous estimons que la description du contexte de conception passe aussi par l'estimation des systèmes de contrainte/astreinte générés par le nouveau dispositif.

2.3.1 Méthode employée

Le recueil a porté sur deux catégories[46] d'informaticiens, à savoir 9 experts en site-central et 8 débutants en client serveur.

Pour déterminer les contraintes générées par ces environnements, nous nous sommes tout d'abord inspirés des taxinomies "autorisées" d'Ombredame et Farverge (1955) ; Leplat et Cuny (1977) et Sperandio (1990). Nous y avons retenu quatre grandes classes de contraintes qui nous paraissaient pertinentes, compte tenu des caractéristiques techniques et organisationnelles des situations de maquettage précédemment établies.
Ce sont les contraintes relatives :

1. à la prise d'information ;
2. aux traitements ;
3. aux savoirs et savoir-faire prérequis ;
4. aux facteurs psycho-sociologiques.

[46] Leurs caractéristiques biographiques ont déjà été exposées lors de la présentation de la "*Démarche méthodologique générale* " (Partie 1.2.4 du Chapitre 3)

Puis, sur la base des protocoles verbaux et des données d'observation recueillis, nous avons cherché à identifier et à ranger les aspects contraignants de l'activité dans chacune de ces rubriques. Etait retenue comme contrainte, toute exigence ayant été formulée par au moins une personne.

La démarche de recueil de données proprement dite s'est déroulée de la façon suivante :

1. d'abord, au cours d'entretiens exploratoires (les mêmes que ceux menés dans la description de l'activité), nous avons demandé aux informaticiens d'énumérer toutes les difficultés, les contraintes, les situations problèmes qu'ils rencontraient dans l'exercice de leur activité. Ce premier "jet" nous a permis de repérer les contraintes les plus fréquentes et/ou les plus difficiles à gérer pour l'informaticien.

2. Ensuite, nous avons soumis cet inventaire à l'épreuve des faits. En d'autres termes, nous avons vérifié si les contraintes décrites lors des entretiens se retrouvaient effectivement dans les situations de travail observées. Nous avons ainsi répertorié toutes les situations critiques dans lesquelles l'informaticien éprouvait soit des difficultés, soit une nervosité dans la réalisation d'une tâche particulière ou dans l'utilisation de l'outil. Nous lui demandions alors d'expliquer les causes de ce malaise. Cette méthode nous a également permis de découvrir de nouvelles contraintes qui n'avaient pas été énoncées lors des entretiens préparatoires.

3. Enfin, au cours d'entretiens complémentaires plus ciblés, les informaticiens devaient réagir aux diverses listes de contraintes que nous avions établies. Concrètement, ils devaient confirmer la présence de ces exigences dans leur activité et indiquer sous quelles conditions elles apparaissaient. Cela a permis d'apporter une validité supplémentaire à ces classements.
   Précisons que 3 experts Pacbase, 5 novices NSDK ont participé à ces entretiens complémentaires.

Des extraits des protocoles verbaux provenant des entretiens exploratoires et ciblés viendront d'ailleurs soutenir et illustrer ces descriptions.

2.3.2 Analyse des contraintes

1) Les contraintes relatives à la prise d'information

Dans cette première catégorie d'exigences, il s'agit de déceler les contraintes de l'environnement qui nécessitent un recueil de données spécifique et/ou une gestion particulière des informations de maquettage.

Ainsi, en passant de l'environnement site-central vers l'environnement client serveur, l'informaticien se trouve confronté à la gestion d'un ensemble de données hétérogènes, provenant de sources diverses (Cf. extrait de corpus). Tandis que dans la conception site-central, toutes les informations utiles à la conception sont centralisées dans une seule source : le dictionnaire.

> *« que cela soit les fonctions réutilisables, les événements à gérer, les contrôles à connaître, l'architecture technique à connaître, c'est tellement diversifié. Il faut aussi savoir le nom de toutes les librairies, du serveur, où il est et ce qu'il comprend comme données. »*

Une autre exigence, commune aux deux environnements, concerne la *réutilisation d'entités informatiques préfabriquées* pour accélérer, simplifier et standardiser les développements. Si cette opération était très facile à réaliser dans le cadre du développement site-central, elle s'avère en revanche plus complexe en client serveur. Les carences des librairies NSDK obligent en effet les informaticiens à créer ces fonctions de toutes pièces ou à les rechercher sur des projets développés par d'autres collègues. Tout cela irrite profondément les informaticiens débutants qui ont l'impression de perdre leur temps et leur énergie dans des tâches inutiles.

> *« Pour la réutilisation, c'est aller rechercher les briques, des composants qui existent quelque part, sans les refaire. C'est donc aller voir à droite ou à gauche, dans d'autres services éventuellement, pour voir si on ne peut réutiliser ce qui a déjà été fait. (...) Chose que l'on faisait nettement moins sur Pacbase, parce qu'on avait un dictionnaire, et les exemples étaient déjà là. »*

	Environnement de conception site-central	Environnement de conception client serveur
Contraintes de gestion de multiples sources d'informations	**Non** : Source d'informations restreinte : Dictionnaire de données unique	**Oui** : Multiple sources d'informations : nombreuses librairies, multiples intervenants…
Contrainte de réutilisation	**Oui** : pour accélérer et simplifier le développement (productivité).	**Oui** : pour optimiser le développement. Toutefois, très peu de fonctions sont disponibles dans ces librairies.

Figure 46
Les contraintes relatives à la prise d'informations

2) Les contraintes relatives aux traitements

Ce deuxième lot de contraintes fait référence à la manière dont le nouvel environnement de travail induit des conduites de conception particulières. On a pu répertorier ainsi des contraintes liées :

a) *à l'organisation de l'activité* : contraintes d'organisation, de simultanéité et de séquentialité ;

b) *aux modalités de résolution du problème* : contraintes de rapidité, de rigueur, de monotonie ;

c) *à la résolution de problème* : contraintes d'opportunité, de créativité, de prévisibilité et de simulation.

a) *les contraintes d'organisation de l'activité*

La *contrainte de simultanéité* paraît être spécifique à la conception client serveur. L'informaticien débutant doit gérer plusieurs tâches parallèlement comme le montre cet extrait de corpus :

> *« on peut être amené durant le coding (codage de la maquette) à gérer en même temps le développement, gérer les versions, le code, se balader dans les spécifications (...) et à penser à plusieurs choses simultanément : le nombre de fonctions à réaliser, les accès à la base, l'architecture technique... à la fin, cela peut être très éprouvant !»*

Les *exigences de séquentialité et d'organisation* –c'est-à-dire de structuration de la tâche– sont à l'inverse peu présentes dans l'environnement client serveur alors qu'elles sont prépondérantes sur le site-central. L'absence des canevas de conception sur NSDK peut expliquer la disparition de ces contraintes :

> *« (...) par rapport à l'organisation de l'activité de maquettage, c'est quand même moins contraignant que sur Pacbase (...)»*

	Environnement de conception site-central	Environnement de conception client serveur
Contrainte de simultanéité	**Non** : séquentialité des tâches	**Oui** : doit s'occuper de l'aspect graphique de la maquette et du comportement de celle-ci
Contrainte de séquentialité	**Oui** : Forte séquentialité de l'activité exigée par Merise, par le cycle de vie en cascade et par les canevas de conception (squelettes de programmation et grille de maquettage)	**Non** : la démarche itérative et incrémentale de développement, ainsi que les outils de conception favorisent un style de conception très souple et empirique
Contrainte d'organisation	**Oui** : besoin de savoir ce qu'il va programmer avant de débuter (compilation coûteuse) + contrainte du squelette qui oblige à programmer selon un ordonnancement précis.	**Non** : environnement plus flexible qui tolère et permet de récupérer les erreurs

Figure 47
les contraintes d'organisation de l'activité

b) les contraintes liées aux modalités de résolution du problème

Une exigence qui semble particulière au client serveur concerne la *rapidité de conception*. Deux raisons à cela :

- d'une part, le client exige plus que par le passé le respect du planning et des coûts annoncés :

  « Avant, on pouvait facilement sortir l'argument : "la technique ne permet pas de le faire". Là, les gens sont peut être plus au courant des choses qui se font, par rapport aux outils de bureautique. Ils nous disent "Dans Word, on peut faire cela, donc c'est facile, et donc vous pouvez le faire". On perd un peu de notre pouvoir ».

- d'autre part, la conception modulaire du projet conduit à la répartition du maquettage entre plusieurs informaticiens. Dans ces conditions, un simple retard de développement peut très vite se propager tout au long de cette chaîne de conception, et rallonger ainsi la production. Une pression constante s'exerce donc sur les informaticiens pour que chacun respecte les plannings annoncés. En conséquence, le concepteur établit des "priorités" de conception :

  « à cause des échéances, les tests de non régression[47] sont par exemple impossibles à réaliser»

[47] Test de non régression : contrôler qu'une modification n'entraîne pas d'erreur sur une autre partie de la maquette ou du programme

Les *contraintes de précision et de rigueur* de conception sont présentes dans les deux environnements informatiques : dans le cas du site-central, cette exigence est due au positionnement dit "relatif"[48] des entités de maquettage sur l'écran (une erreur de quelques centimètres décale toutes les autres rubriques situées à sa périphérie). Dans le cas de NSDK, la contrainte provient de la nécessité de *i)* sélectionner les objets graphiques appropriés ; *ii)* de les organiser convenablement sur la fenêtre ("*au pixel près*", selon la charte graphique en vigueur à DABFI-O), et *iii)* de programmer judicieusement les événements à associer à ces objets, tout en évitant de possibles incohérences.

> *« La normalisation de la conception graphique* [par la charte ergonomique, NDR] *et de la programmation* [déportation du code dans une librairie générale] *nous imposent d'être plus précis et consciencieux que sur Pacbase.»*

En revanche, la *contrainte de monotonie* semble avoir disparu. Cette exigence que l'on trouve sur le site-central fait référence à la répétition d'une même tâche ou d'une même procédure ainsi qu'aux faibles marges de manoeuvre laissées dans le travail. En effet, si le paradigme site-central impose un modèle de conception récurrent auquel l'informaticien peut difficilement échapper, l'environnement client serveur se singularise par sa grande latitude d'action, avec des tâches originales et diversifiées.

> *« quand on fait nos 50 maquettes avec Pacbase ; ne faire que ça, cela peut devenir rapidement rébarbatif. Un écran appel toujours un écran, on dispose toujours les rubriques de la même façon, etc. »*

	Environnement de conception site-central	Environnement de conception client serveur
Contrainte de rapidité	**Non** : Le programmeur réalise pratiquement seul l'ensemble de la maquette (conception monolithique). Son rythme de travail n'affecte donc guère celui des autres développeurs	**Oui** : Conception modulaire (partage des tâches) : tout retard se répercutera sur le rythme de travail des autres. De plus, le client exige le respect des délais
Contrainte de précision et de rigueur	**Oui** : dans la définition des paramètres des entités de maquettage (maquettage relatif)	**Oui** : pour la sélection et la programmation des objets graphiques et de leurs événements, pour la présentation normalisée de l'interface, pour la dispersion du code…
Contrainte de monotonie	**Oui** : les contraintes graphiques et logiques de l'environnement site-central font que la conception des maquettes reste souvent identique.	**Non** : très grande latitude dans la conception de l'interface.

Figure 48

les contraintes liées au mode de résolution du problème

c) Les contraintes liées à la résolution du problème

[48] Positionnement relatif d'une rubrique sur un écran site-central : l'informaticien positionne une nouvelle rubrique en prenant comme point de repère celle qui lui est immédiatement adjacente.

Trois nouvelles contraintes apparaissent avec l'utilisation de l'environnement client serveur :

- Les contraintes *d'opportunité et de flexibilité*, c'est-à-dire que l'informaticien doit être capable de changer à tout moment de plan de maquettage dès qu'une solution meilleure apparaît ou que l'utilisateur exprime un besoin particulier. Cette exigence de plasticité n'est pas caractéristique de la conception site-central puisque ses canevas prescrivent une démarche qui reste inamovible, quelles que soient les circonstances du développement.

  « Plus on rentre dans le détail du maquettage [en client serveur, NDR], *plus on se pose de nouvelles questions, et plus on trouve de nouvelles pistes qu'il faut essayer d'intégrer et de tester (...) A tout moment le client peut nous demander d'ajouter une fonctionnalité graphique parce qu'il l'a vue sur un logiciel bureautique. On doit alors s'adapter »*

- Les contraintes de *créativité* car la quasi-absence de ressources préexistantes oblige l'informaticien débutant à les imaginer intégralement.

  « c'est toujours plus difficile de créer quelque chose qui n'existe pas que de s'inspirer ou de modifier quelque chose qui existe déjà comme les écrans de Pacbase »

- Les contraintes de *prévisibilité et d'anticipation* car l'informaticien doit non seulement s'assurer de la cohérence entre les différents événements de l'interface, mais il doit aussi contrôler leurs effets sur les (futurs) composants de la maquette.

  « On devra toujours se demander ce qui peut arriver si l'on ajoute ou si l'on retire telle commande ou tel événement. Rajouter un objet peut changer le comportement de la maquette»

  Sur ce dernier aspect, on notera que l'anticipation apparaît comme une réponse pour faciliter la gestion de la dynamique temporelle d'un processus. En effet, l'anticipation permet de se préparer à agir, d'actualiser ses connaissances sur le système et de gagner du temps dans certains cas.

- La contrainte de *simulation* est commune aux deux environnements. Toutefois, la vocation de ces tests diffère d'un contexte technique à l'autre : en site-central, la simulation sert principalement à la visualisation des écrans après leur paramétrage sur la grille de maquettage : l'informaticien peut ainsi apprécier l'organisation de l'écran et corriger si besoin est. Tandis qu'en client serveur, la simulation est employée pour tester les comportements de l'interface.

  « il y a un gros besoin de simulation, pour voir si la maquette passe correctement dans tous les cas. S'il n'y a pas une combinaison qui peut fausser. Il faut absolument tester tous les cas [en client serveur, NDR] *»*

	Environnement de conception site-central	Environnement de conception client serveur
Contrainte d'opportunité et de flexibilité	**Non** : à cause de l'environnement figé.	**Oui** : à cause des modifications et des évolutions de versions exigées par le client.
Contrainte de créativité	**Non** : reprend et développe des modèles préexistants	**Oui** : peu ou pas de modèles de fenêtres disponibles
Contrainte de prévisibilité et d'anticipation	**Non** : la logique de fonctionnement de la maquette est toujours la même : un écran appelle un autre écran.	**Oui** : à cause de la gestion cohérente et prévisionnelle des événements.
Contrainte de simulation	**Oui** : lors de la conception de la maquette, il faut régulièrement simuler l'écran réalisé (simulation informatique). A cela s'ajoute une simulation mentale pour représenter la disposition de l'écran lors du codage sur la grille.	**Oui** : pour tester l'aspect mais aussi et surtout le comportement de l'interface (événement).

Figure 49
Les contraintes liées à la résolution du problème

3) Les contraintes relatives aux savoirs et aux savoir-faire

Troisième classe de contraintes, celles qui font référence aux savoirs (connaissances théoriques) et aux savoir-faire prérequis (expériences) pour utiliser convenablement le nouvel environnement de conception.

On peut ainsi remarquer que si Pacbase exigeait la rétention d'un très grand nombre de commandes (plus de 300 recensées), NSDK allège considérablement cet effort de mémorisation en proposant un accès direct aux commandes par des menus, des boutons et autres représentations iconiques.

« C'est quand même plus cool sur NSDK car il n'y a pas toutes les commandes à se rappeler. On prend la souris, et puis "clique", on a ce que l'on veut »

En plus de la connaissance des commandes de NSDK, l'informaticien doit acquérir des *compétences techniques* relatives à l'architecture client serveur afin d'améliorer les performances des applications élaborées.

D'un point de vue fonctionnel, le nouveau cycle de vie par prototypage exige que l'informaticien s'investisse dans toutes les phases du processus de conception –de l'analyse à la programmation–. Cette polyvalence nécessite le recours à des savoirs plus larges et originaux.

> *« Sur site central, tout est central. Il n'y a donc pas de questions d'architecture et de réseau à se poser. Sur client serveur, par contre, il va y avoir des choix à faire. (...) chaque personne devra être plus technique. »*

De même la *représentation de l'utilisateur final* est devenue une donnée fondamentale de la conception client serveur. Ce regain d'intérêt s'explique par le fait que l'informaticien doit désormais élaborer des maquettes qui sont compatibles avec la logique d'utilisation de l'opérateur :

> *«A l'opposé de Pacbase où l'on se représentait toujours le même utilisateur final standard, quelle que soit l'application que l'on développait ; sur NSDK on doit avoir en tête à chaque nouveau projet un profil d'utilisateur différent, pour adapter le plus possible le comportement de l'interface à la conduite de cet utilisateur »*

Enfin, une dernière contrainte nécessite de *documenter* convenablement la maquette élaborée. Cette obligation répond à l'organisation modulaire de la conception. En effet à cause des nombreuses personnes impliquées dans le projet, et de la complexité de la phase de programmation, chaque informaticien rédige une documentation très précise des modules qu'il élabore. Ces notes se révéleront très utiles lors de la recomposition finale de la maquette ou durant les phases de maintenance applicative. Mais elles présentent aussi un tout autre intérêt dans la mesure où les informations qu'elles recèlent peuvent renseigner d'autres informaticiens sur l'existence de fonctions informatiques potentiellement réutilisables. En somme, cette documentation permet de combler les lacunes des librairies techniques de NSDK.

> *« la doc est très utile parce qu'on peut faire des recherches, comprendre à quoi une fonction sert, ce qu'elle fait. Cela est nécessaire pour optimiser le développement »*

	Environnement de conception site-central	Environnement de conception client serveur
Contrainte liée à la manipulation de l'environnement	**Oui** : Nombreuses codifications et commandes à retenir	**Non** : manipulation directe, boutons, menus…(soulage la mémoire)
Contrainte de polyvalence et de polycompétence	**Non** : Informaticien spécialisé sur un domaine (analyse ou programmation) : connaissances spécifiques	**Oui** : Informaticien intervient sur plusieurs domaines et doit connaître les arcanes de l'architecture technique : polycompétence requise
Contrainte de représentation (de l'utilisateur final)	**Non** : l'utilisateur est peu pertinent car la maquette est construite exclusivement sur un modèle technique qui ne peut être adapté.	**Oui** : La représentation de l'utilisateur est fondamentale car l'application doit être pensée, conçue et programmée en fonction de l'utilisateur final
Contrainte de documentation	**Non** : le périmètre d'action n'est pas suffisamment fluctuant pour recourir à de telles données	**Oui** : les informaticiens travaillent à plusieurs sur l'application et échangent des modules techniques. La documentation doit faciliter la lecture de ces parties

Figure 50
Les contraintes relatives au savoir et au savoir-faire

4) **Contraintes relatives à des facteurs psycho-sociologiques**

Dernier groupe de contraintes, celles qui requièrent des conduites sociales particulières de la part de l'informaticien.

Ainsi, les *contraintes de concertation, négociation et de recherche de compromis* sont plus importantes en client serveur que sur le site-central. En effet, l'informaticien ne peut plus imposer ses orientations techniques ; il doit dorénavant consulter et négocier avec l'utilisateur final pour toutes les décisions relatives au projet. Cette contrainte marque la fin d'une période caractérisée par la domination de l'informaticien sur l'utilisateur.

> *« On doit trouver des accords, des terrains d'entente avec tous ceux qui interviennent et jugent le projet. On doit négocier aussi avec les utilisateurs qui ont certainement pris un pouvoir plus conséquent qu'auparavant »*

De la même manière, *les contraintes d'échange, d'entraide et de bonne entente* révèlent l'extrême dépendance de l'informaticien vis-à-vis de son environnement professionnel, et en particulier des nouveaux acteurs de la conception client serveur. Chacun se voit propulser responsable d'un domaine particulier de développement (gestion des bases de données, du réseau et de l'architecture, des librairies…). Le maintien de relations sinon cordiales, du moins professionnelles, avec ces nouveaux partenaires est le prérequis nécessaire pour suivre le déroulement du projet, ou alors, pour espérer compter sur l'aide de ces spécialistes en cas de problèmes techniques.

« Je pense que les communications sont plus importantes parce que toutes les fonctions sont liées. On est obligé de beaucoup communiquer pour se dire : moi je fais ça là, je vais modifier ça, et je suis le seul à le modifier actuellement. Alors qu'en site central, c'était moins comme cela »

	Environnement de conception site-central	Environnement de conception client serveur
Contraintes de concertation, de négociation, de recherche de compromis.	**Non** : lors des discussions avec les clients, les arguments techniques de l'information l'emportent presque toujours (pouvoir du côté informatique)	**Oui** : face à la prise de pouvoir de l'utilisateur, l'informaticien doit développer d'autres argumentaires (coût du développement, performance du système, délai de livraison plus important)
Contraintes de communication, d'entraide, d'échange et de "bonne entente".	**Non** : homogénéité de la conception limite le besoin d'aide extérieure	**Oui** : la conception implique de nouveaux partenaires qui sont porteurs d'informations nécessaires au développement.

Figure 51
Contraintes relatives aux facteurs psycho-sociologiques

2.3.3 Analyse des astreintes

Face à ces nombreuses contraintes, l'informaticien peut ressentir une surcharge mentale[49]. Sur le versant psychologique, le stress, la sensation d'épuisement physique ou mental ou encore le développement de troubles psychosomatiques pourront être les symptômes d'une charge mentale excessive. Sur le versant social, les résistances au changement, les critiques envers la hiérarchie, l'irritation, la démotivation seront les réponses que l'individu pourra adopter face à des exigences trop fortes.

Deux groupes d'astreintes ont été déduits à partir des données recueillies lors de nos entretiens et observations. Cette méthode d'évaluation subjective des astreintes est la seule qui a pu être réalisée, compte tenu des conditions d'intervention fixées par la hiérarchie : refus de la passation d'un questionnaire et du test de la double tâche.

1. Un premier type d'astreinte correspond à la *surcharge cognitive* générée par les nombreux traitements effectués pour interpréter et gérer des informations disparates : recherche de l'information pertinente parmi un ensemble de données, simulation du comportement des objets, évaluation des "effets globaux"... sont

[49] La surcharge mentale intervient lorsque les sollicitations de l'environnement de travail (c'est-à-dire les contraintes) sont supérieures aux ressources cognitives de l'individu. On parlera de "sous-charge" lorsque les sollicitations sont inférieures aux ressources.

autant d'opérations qui réclament la mobilisation d'un ensemble d'aptitudes que le débutant NSDK maîtrise encore difficilement. De même, la programmation de la maquette est à l'origine d'une fatigue nerveuse importante par le niveau de vigilance qu'elle requiert. Cette phase impose une attention soutenue pour à la fois s'occuper de plusieurs tâches concurrentes (*contrainte de simultanéité*), travailler sous la pression du temps (*contrainte de rapidité*), développer avec rigueur (*contrainte de précision*) mais aussi avec une certaine souplesse dans la gestion de la tâche (*contrainte d'opportunité*).

> *«Lorsque je veux savoir quelles fonctions sont appelées et éditer directement le code, je suis obligé de savoir dans quelles librairies elles sont et connaître éventuellement le chemin d'accès. Quand tu le répètes souvent, c'est pénible. Tu vas dépenser du temps et de l'énergie à te souvenir du chemin, à le taper (...) A la limite, tu vas perdre ton idée initiale»*

2. Un second type d'astreinte concerne l'image que l'informaticien se fait de sa place dans le système de travail, c'est-à-dire du type de reconnaissance professionnelle qu'il attribue à sa fonction. Le développeur a ainsi de plus en plus de mal *(i)* à définir sa contribution exacte à un processus développement collectif (conception modulaire), *(ii)* à déterminer son territoire professionnel et à affirmer ses compétences dans un groupe de travail qui s'enrichit de nouveaux acteurs. De même, certaines décisions ou responsabilités qui lui étaient traditionnellement réservées sur le site-central, lui échappent et sont désormais assumées par les spécialistes client serveur et les utilisateurs finaux. C'est pourquoi, d'aucuns s'interrogent sur la reconnaissance de leurs compétences dans un contexte professionnel qui devient de plus en plus concurrentiel :

> *« On n'a pas tellement de répondant par rapport à ce qu'ils* [les prestataires, NDR] *nous proposent, du fait de notre méconnaissance(...) C'est très dur de se dire que l'on dépend de ces personnes de l'extérieur, alors que nous étions bons sur l'ancien environnement. Les rôles sont inversés. Je passe la main au prestataire qui s'y connaît mieux que moi. C'est frustrant d'être derrière, et de regarder à son tour »*

2.3.4 Synthèse - Discussion

Dans cette analyse, nous avons comptabilisé onze nouvelles contraintes et deux types d'astreinte possibles. Ce double constat nous amène à développer deux niveaux de réflexion :

UNIVERSITÉ DE METZ BU Lettres S C D

- D'une part, quels sont les nouveaux comportements professionnels et les nouveaux apprentissages que l'informaticien doit mettre en œuvre pour gérer ces contraintes ?
- D'autre part, les astreintes risquent-elles de contrarier le processus d'appropriation du nouvel environnement ?

a) Compétences à acquérir pour gérer ces nouvelles contraintes

Ce tableau (Cf. Figure 52) répertorie les onze nouvelles contraintes apparues avec le nouvel environnement et dresse la liste des compétences requises pour les maîtriser.

TYPE CONTRAINTES GÉNÉRÉES PAR LE NOUVEL ENVIRONNEMENT	COMPÉTENCES REQUISES
1. Contrainte de gestion de multiple sources d'informations	– Attention partagée – Catégorisation – Mémorisation – Analyse synthétique
2. Contrainte de simultanéité	– Plasticité cognitive – Attention partagée
3. Contrainte de rapidité	– Maîtrise des stratégies de résolution – Expérience préalable requise – Recours à des automatismes de conception
4. Contraintes d'opportunité et de flexibilité	– Plasticité cognitive – Adaptabilité des schémas de résolution – Réactivité
5. Contrainte de créativité	– Heuristiques – Imagination créatrice – Stratégie de résolution de problèmes nouveaux
6. Contraintes de prévisibilité et d'anticipation	– Simulation mentale – Maîtrise parfaite des schémas de résolution
7. Contrainte de polyvalence	– Multi-compétences sur des savoirs et des savoir-faire diversifiés de la conception
8. Contrainte de représentation (de l'utilisateur final)	– Méthodes d'analyse de l'activité – Modèle conceptuel de l'utilisateur et de sa tâche – Implémentation dans un modèle informatique
9. Contrainte de documentation	– Compétences rédactionnelles
10. Contraintes de concertation, de négociation, de recherche de compromis.	– Compétences sociales – Explicitation de savoirs et savoir-faire
11. Contraintes de communication, d'entraide, d'échange et de "bonne entente".	– Compétences communicationnelles – Approche gestionnaire et sociale de la conception

Figure 52

Tableau récapitulatif des contraintes et des compétences requises

Finalement, trois grandes classes de compétences se dégagent de cet inventaire:

- des compétences cognitives originales pour la résolution des problèmes nouveaux, l'ajustement de la solution aux circonstances de la conception et la gestion de données multiformes ;

- des compétences sociales basées sur une approche communicationnelle de la conception et sur l'explicitation des savoirs et savoir-faire ;
- des compétences plus empiriques et pragmatiques liées à la connaissance de l'utilisateur et de sa tâche.

b) Les incidences des astreintes sur l'appropriation du nouveau système

En ce qui concerne ces incidences, il semble que le processus de migration technologique ne soit pas remis en cause. En effet, la plupart des informaticiens que nous avons interrogés perçoivent la transition vers le client serveur comme une véritable aubaine, leur donnant l'occasion d'acquérir de nouvelles compétences et, surtout, de pouvoir se repositionner sur un marché professionnel en pleine mutation.

> *« Les techniques évoluent très vite. Et la norme d'ici 3-4 ans va être le client serveur. Si on reste sur site-central, on va être à la traîne. De ce point de vue, c'est donc une nécessité de migrer»*

D'autres voient dans l'accession à cette technologie le moyen de rompre avec l'utilisation monotone et aliénante de Pacbase. Ils fondent l'espoir de retrouver une certaine liberté et émulation intellectuelle par la découverte d'un environnement de travail plus évolué:

> *« Sur Pacbase, on est vraiment dans notre monde, et on n'en bouge pas. On n'en sort pas. On ne se pose même plus la question à savoir comment gérer telle ou telle fonction (...) j'attends avec impatience NSDK car cela fait 4 ans que je travaille sur Pacbase ; cela devient routinier.»*

Enfin, il y a les personnes résignées. Elles savent qu'à défaut de faire l'effort de s'intégrer au changement, elles encourent le risque de stagner fonctionnellement et techniquement sur le site-central, dans des postes à faible valeur ajoutée (de maintenance principalement).

> *« La plus grande crainte des gens est qu'on ne les choisisse pas pour travailler sur client serveur. En fait, on ne s'est pas posé le problème en forme de craintes de nouvelles techniques, mais crainte de rester sur les anciennes techniques. »*

En somme, les informaticiens abordent la migration technique avec un certain enthousiasme. Les contraintes, pourtant nombreuses, ne semblent pas les affoler outre mesure. En fait, à travers la maîtrise de ces nouvelles techniques client serveur, les informaticiens débutants souhaitent progresser vers une nouvelle identité et reconnaissance professionnelle :

> *« Je dirais que c'est le prix à payer pour ne pas devenir archaïque »*

2.4 CONCLUSION GÉNÉRALE SUR L'ANALYSE DU SYSTÈME HOMME-MACHINE

La description des environnements de conception site-central et client serveur nous a permis de dresser un tableau assez exhaustif des caractéristiques techniques et organisationnelles propres à chaque environnement de travail.
Les comparaisons effectuées ont notamment montré que si des spécificités notables les opposaient, exigeant par là même, la remise en cause de certaines pratiques professionnelles, il préexistait néanmoins des caractéristiques communes qui pouvaient rapprocher ces environnements et servir opportunément de passerelles pour des transferts d'apprentissage. En particulier, ce sont les aspects qui touchent aux instructions de programmation, aux paradigmes du code généré par les outils (procédural), aux dispositifs de stockage des entités de programmation (bibliothèque de Pacbase et librairie de NSDK), aux techniques de maquettage (réalisation dynamique de Pacbase et positionnement graphique de NSDK) et aux bases de données (DB2 et Sybase).

Cela dit, ces convergences restent assez limitées par rapport aux nombreuses innovations techniques. On dénombre plus de ruptures technologiques entre les deux environnements que de convergences. Dans ces conditions, les informaticiens débutants doivent développer deux types de conduite : multiplier les nouveaux apprentissages pour appréhender les caractéristiques originales du client serveur tout en minimisant le recours aux anciennes connaissances du site-central. En effet, l'informaticien, comme tout être humain, est naturellement enclin à se tourner vers les ressources cognitives dont il dispose le plus facilement ; ici celles du site-central. Ces réutilisations représentent une économie cognitive non négligeable dans un contexte d'apprentissage intensif. Toutefois, on verra plus loin la gageure d'une telle entreprise car les transferts négatifs qui en résultent peuvent être à l'origine d'erreurs de programmation qui nécessiteront le redéploiement de nouvelles procédures de correction. Et finalement, le gain cognitif obtenu par la réutilisation peut très bien disparaître par la mise en œuvre de stratégies de récupération supplémentaires.

Au niveau organisationnel, on a relevé de nombreux changements dans la conduite de l'activité de maquettage, en particulier :

- une gestion empirique de l'activité, affranchie de tout carcan technique : il ne s'agit plus de s'insérer dans un programme pré-construit et de remplir des zones d'une grille prédéfinie, mais de construire *ex-nihilo* des scénarios de maquettage ;
- une véritable démarche de création de solution de maquettage où les simulations sont fréquentes et déterminantes pour le processus de résolution ;
- un élargissement des domaines d'intervention couverts par l'informaticien : il n'est plus cantonné seulement à la réalisation de la maquette, mais intervient aussi en amont, lors des phases d'analyse et de recueil des besoins des utilisateurs.

C'est donc une tâche plus laborieuse et moins systématique que sur Pacbase, où les tâtonnements de conception et les simulations se succéderont sur la base d'un processus itératif. De même, là où il n'y avait qu'une seule procédure applicable, l'informaticien client serveur se retrouve à gérer plusieurs combinaisons de solutions admissibles. Aussi, malgré les libertés que consent NSDK dans l'élaboration de la maquette, les informaticiens doivent se discipliner au risque de se perdre très rapidement dans les méandres de la programmation événementielle et graphique. Nous aurons l'occasion de revenir sur ce point lors de l'analyse cognitive du maquettage.
On assiste également à l'avènement d'une conduite de projet plus participative et collective, impliquant ainsi une approche plus gestionnaire et sociale du développement.

Enfin, l'environnement client serveur est également le vecteur de nouvelles contraintes (de rapidité, de rigueur, de simultanéité des traitements ; de gestion de données hétérogènes, d'opportunité, de créativité...). Ces exigences obligent l'informaticien à développer de nouvelles compétences et de nouvelles aptitudes de travail (anticipation, plasticité, flexibilité, réactivité, créativité, sociabilité,...). Elles provoquent aussi une augmentation de sa charge mentale par les astreintes générées. Néanmoins, ces contraintes ne semblent pas constituer une menace pour la migration

technologique, tant le désir de changer d'outil et d'évoluer vers de nouvelles fonctions professionnelles est grand.

En définitive, toutes ces évolutions se caractérisent par de nouvelles formes d'articulation entre l'individu, le système et son activité. La tâche elle-même se modifie (la prescription informatique évolue vers une définition par objectifs, de nouvelles tâches de diagnostic et de coordinations apparaissent) tandis que les conditions d'exercice de l'activité se transforment (les repères disparaissent, les modes d'analyse et de résolution deviennent plus globaux). Le débutant doit quant à lui faire l'effort de renoncer à d'anciennes structures de connaissances obsolètes pour acquérir les compétences idoines. Dès lors, de nouveaux compromis cognitifs sont à construire par ces informaticiens dans des situations de plus en plus aléatoires et de moins en moins répétitives. C'est à l'analyse de ces processus de raisonnement que le second chapitre est consacré.

3. L'ANALYSE COGNITIVE DE L'ACTIVITÉ DE MAQUETTAGE

L'objectif de cette seconde partie est de sérier les principales conduites cognitives mises en œuvre par les informaticiens dans chaque environnement de conception, et de rendre compte des phénomènes de transfert d'apprentissage qui se déroulent lors de la migration technologique. L'idée est de montrer que les difficultés rencontrées par les informaticiens novices dans l'apprentissage d'un nouveau dispositif ou encore les nombreuses erreurs qu'ils peuvent commettre dans leur nouvelle tâche de conception proviennent de l'incompatibilité des compétences récupérées du domaine source avec celles requises pour maquetter dans le domaine cible.

L'analyse de ces compétences passe par la modélisation de l'informaticien dans chaque contexte de conception. C'est précisément l'objectif de cette partie, où par l'utilisation de notre modèle d'analyse, nous présenterons successivement :

1) les démarches de structuration de l'activité (modèle d'action),

2) les stratégies de résolution mises en œuvre (modèle de résolution)

3) la qualité de la relation homme-machine (modèle d'interaction),

4) les connaissances et les trames de référence employées pour concevoir les interfaces (modèle de référence).

3.1 LE MODÈLE D'ACTION

3.1.1 Méthode employée

a) Rappel de l'hypothèse

Selon nos hypothèses, les informaticiens auraient deux possibilités pour conduire leur activité : *(i)* soit ils suivent scrupuleusement le plan de maquettage qu'ils ont planifié ; *(ii)* soit ils dévient opportunément de celui-ci. Ces contingences d'actions dépendraient des degrés de liberté qu'octroient les environnements de conception. En termes clairs, l'environnement site-central –qui a été présenté comme prescriptif dans les parties précédentes– imposerait une organisation hiérarchisée de la tâche qui ne saurait être remise en cause ; tandis que l'environnement client serveur, décrit comme plus souple et adaptable, favoriserait une gestion plus circonstancielle de l'activité.
Dès lors, les problèmes rencontrés par les novices NSDK proviendraient de la difficulté à s'émanciper fonctionnellement, c'est-à-dire de se libérer de la tutelle technique qu'exercent les canevas de Pacbase sur leur activité.

b) Techniques de recueil de données

Ce sont d'abord des *entretiens exploratoires* réalisés auprès d'un échantillon de 9 experts Pacbase, 7 experts NSDK et 8 débutants NSDK. Nous avons demandé à ces personnes de décrire leur activité et de préciser l'influence que le système technique pouvait avoir dans la conduite de leur tâche.
Les descriptions fournies par deux personnes assurant la formation et l'assistance technique sur les deux systèmes Pacbase et NSDK ont également été prises en compte. De même, toute la documentation relative à la composition d'un projet informatique a été consultée.

L'observation de l'activité réelle nous a conduit à relever toutes les données et tous les éléments qui montraient comment l'informaticien organisait son travail : par exemple, la représentation graphique des fenêtres sur le papier, la consultation de "bibles" papiers où sont stockés les anciens projets, les discussions entre collègues pour connaître l'existence d'une fonction pouvant être réutilisée, etc. Nous avons noté tous les éléments qui pouvaient suggérer une remise en cause partielle ou totale du plan : par exemple, les manifestations bruyantes de l'informaticien lorsqu'une idée de

conception ne donnait pas le résultat escompté, les critiques des collègues ou des experts, la fréquence d'utilisation de la fonction de simulation...
Ces observations ont touché 5 informaticiens de chaque catégorie durant une période de 60 minutes en moyenne.

La technique de *verbalisation provoquée*[50] a elle aussi été employée : nous demandions aux informaticiens de penser à haute voix, c'est-à-dire de commenter les actions à mesure qu'il les réalisait. Cette méthode permet d'inférer les processus cognitifs en œuvre durant l'activité. Ce recueil s'est déroulé durant la phase d'observation, et a impliqué 5 informaticiens de chaque catégorie. Il portait sur une tâche de maquettage (réalisation d'une fenêtre ou d'un écran) et durait 30 minutes en moyenne.

c) Indicateurs retenus et méthode d'analyse

Sur la base de ces recueils, trois types d'indicateurs ont été retenus pour déterminer l'organisation de l'activité. Il s'agit :

1. *du nombre de fois où l'informaticien signalait une quelconque influence de l'environnement technique sur la structuration de son activité.*
   Ces descripteurs ont été obtenus par une analyse thématique effectuée sur les corpus des entretiens exploratoires. Ils donnent une première idée sur la manière dont les informaticiens perçoivent le niveau de contrôle des systèmes techniques sur le déroulement de leur tâche.

2. *de la décomposition de la tâche de maquettage en une structure de sous-buts et de la comparaison de cette matrice avec les conduites effectives des informaticiens.*
   Cette représentation de la tâche prescrite en parties élémentaires et hiérarchisée correspond à une sorte de notice de maquettage. Pour élaborer cette structure, nous avons créé notre propre formalisme en nous inspirant de GOMS[51] et du modèle de représentation hiérarchique de la tâche (Sebillote, 1991), où :

[50] Cf. "*Annexe III*" pour une présentation détaillée de cette méthode. On trouvera aussi une discussion plus générale qui porte sur la validité des techniques de verbalisation pour recueillir l'expertise et les compétences d'un individu.
[51] Les justifications de l'utilisation du formalisme GOMS sont exposées en "*Annexe VI*".

a) *Goal* correspond à l'objectif général ou l'état final de maquettage à atteindre ;
b) *Sub-Goal* sont les sous-buts intermédiaires ou les prérequis par lesquels il faut nécessairement passer avant d'atteindre l'objectif final ;
c) *Opérators* : désigne les procédures et les séquences d'action à mettre en œuvre ;
d) *Méthods* : indique les moyens (techniques, fonctionnelles) utilisés pour réaliser ces procédures ;
e) *Selection Rules* sont les règles d'action de type : "*Si... alors... ; sinon...*"

Concrètement, sur la base des entretiens exploratoires réalisés auprès des experts (informaticiens et formateurs) et des informations prélevées dans la documentation, nous avons utilisé notre formalisme pour composer les structures de buts générales de chaque tâche de maquettage (sur Pacbase, puis pour NSDK). Ensuite, ces représentations ont été exploitées comme des grilles d'analyse des conduites individuelles : les données obtenues sur le terrain (par verbalisation provoquée et observation) étaient systématiquement comparées à ces représentations hiérarchiques. En d'autres termes, nous confrontions tâche prescrite et activité réelle, et expliquions les écarts entre ces deux modèles. Des extraits des entretiens sont utilisés pour soutenir et illustrer nos analyses.

3. *de la capacité de l'informaticien à modifier ou à revenir sur son plan*

Ce dernier indicateur rend compte des possibilités techniques qu'offre chaque environnement pour corriger les plans d'action. Nous avons également comptabilisé le nombre de simulations réalisées durant la phase de maquettage, en tant qu'elles renvoient aux tests effectués sur les plans de maquettage, et qu'elles peuvent donc conduire à la remise en cause de ce plan.

3.1.2 Présentation des résultats

Ce tableau présente les résultats obtenus par chaque échantillon.

ELÉMENTS DE COMPARAISON	EXPERTS PACBASE	EXPERTS NSDK	NOVICES NSDK
PERCEPTION DU NIVEAU DE CONTRÔLE EXERCÉ PAR LES SYSTÈMES TECHNIQUES SUR L'ACTIVITÉ (indiqué par le nombre d'items moyen exprimé par informaticien)			
Moyenne d'items indiquant une structuration de l'activité par l'environnement (obtenue à partir des entretiens exploratoires) Exemples d'items : • *" un déroulement imposé (...) une organisation de l'activité imposée",* • *"structuration du travail",* • *"ossature de conception à suivre",* • *"contraintes fortes de l'environnement",* • *"forcer à suivre les indications du système "* • *"prescriptions importantes".* • *pas de retour en arrière possible, etc.*	*32* (Interventions faisant toutes références au poids de Pacbase et de ses canevas dans la conduite de l'activité)	*4* (Interventions soulignant que NSDK était plus contraignant que d'autres L4G ; notamment en ce qui concerne la concentration du code dans une librairie unique)	*20* (Interventions pointant sur l'absence de structuration de l'activité par l'environnement)
Moyenne d'items indiquant une mise en œuvre autonome et personnelle de l'activité (obtenus à partir des entretiens exploratoires). Exemple d'items: • *"Liberté de travail" "Faire ce que l'on veut"* • *"possibilité de tout faire",* • *"Souplesse", "flexibilité",* • *"commencer, stopper, revenir en arrière, recommencer sans problème",* • *"absence de contraintes (...) de contrôle"* • *"Ouverture complète", "Indépendance"*	*5* (Les plus anciens et les plus expérimentés soutiennent qu'il est possible de transgresser son canevas par du codage spécifique)	*15*	*40* (ce grand nombre d'occurrence souligne la nouvelle liberté acquise dans l'environnement client serveur)
CARACTÉRISTIQUES DES PLANS ÉTUDIÉS			
Structures de buts de chaque tâche Moyenne des sous-buts identifiés Par exemple : • *"je vais rappeler un vieil écran, le modifier puis le tester"* (= 3 sous-buts sur Pacbase) • ou *"je vais installer un bouton dans une fenêtre, puis appeler un événement, ensuite je le coderai et le testerai"* (=5 sous-buts sur NSDK)	10	14	
Capacité à changer de plan grâce aux possibilités techniques de l'environnement	*Faible* (rigidité des canevas de conception, techniques de simulation peu performantes et coût de simulation élevé à cause du langage compilé)	*Importante* (Souplesse de l'environnement, langage interprété favorisant l'expérimentation, fonction de simulation performante)	
Nombre moyen de simulations effectué durant la phase de maquettage	*1* (en toute fin de maquettage)	*13* (en fin de conception des fenêtre)	*25* (après chaque programmation d'événements)

Figure 53

Tableau récapitulatif des résultats du modèle d'action

Si l'on se réfère aux commentaires des informaticiens, on se rend compte que le contrôle de l'activité par le système est davantage ressenti du côté des informaticiens site-central (*32 références en moyenne*) que du côté des développeurs client serveur (*4*

références pour les experts et 20 pour les novices). Mais dans ce dernier cas, les informaticiens débutants insistent surtout sur l'absence de structuration de l'activité par NSDK, et comparent cette nouvelle tâche avec les conditions particulières de leur ancienne activité site-central.

Par ailleurs, le nombre moyen de sous-buts déterminé dans les structures de buts est plus élevé dans les échantillons NSDK (14 *sous-buts*) que dans celui de Pacbase (10 *sous-buts*). Ces résultats sont surprenants car, d'après nos prévisions, nous nous attendions plutôt au rapport inverse ; c'est-à-dire davantage de sous-buts en site-central qu'en client serveur à cause des contraintes techniques. En effet, un nombre élevé de buts serait symptomatique d'une organisation hiérarchisée de l'activité (Sebillote, 1988). Nous reviendrons plus tard sur ces données paradoxales.

On remarque également que la capacité à modifier le plan au cours de l'activité est plus importantes sur NSDK qu'avec Pacbase. Cela tient aux dispositifs techniques qui, dans un cas, supervisent l'activité (canevas de programmation de Pacbase) et, dans l'autre, favorisent les déviations opportunistes du plan (fonction de simulation de NSDK). Le nombre moyen de simulations réalisé durant le maquettage (plus d'une vingtaine) atteste d'ailleurs de la facilité avec laquelle ces tests peuvent être réalisés. Mais ce chiffre peut également témoigner du niveau de confiance que les informaticiens accordent à leur plan de résolution : plus l'individu doute de la fiabilité de ses solutions, plus il voudra les tester afin de vérifier leur efficacité. C'est sans doute pour cette raison que les novices effectuent en moyenne 2 à 3 fois plus de simulations que les experts NSDK, et plus de 20 fois plus que les experts Pacbase.

En définitive, ces résultats tendraient à prouver que l'environnement de conception Pacbase du site-central, *via* ses canevas de conception, conditionnent la planification de l'activité de manière hiérarchisée. En outre, s'il existe un plan d'action chez les informaticiens client serveur, celui-ci évoluerait opportunément durant l'activité de maquettage. Nous allons à présent examiner plus en détails tous ces indicateurs, en revenant notamment :

1. sur la comparaison entre la structure de buts de la tâche et les conduites observées ;
2. sur les possibilités techniques que l'informaticien a ou non de modifier ses plans d'action.

3.1.2.1 La planification dans Pacbase

3.1.2.1.1 Structure de buts définie pour la tâche de maquettage sur Pacbase

La structure de buts de la tâche de maquettage rend compte de deux types d'action :

1. l'élaboration d'un plan d'action général durant l'analyse organique
2. l'application de ce plan d'action au cours de la phase de maquettage

A) Goal :		- ***Elaborer un plan d'action de maquettage durant l'analyse organique***
va induire différents sous-buts :		
Sub-Goal 1		- Désigner les rubriques à récupérer dans le dictionnaire.
Operator :		- Vérifier si elles existent et si elles conviennent, et les désigner sur le document de maquettage.
Methods		- Fonction de recherche de rubrique dans le dictionnaire.
	Selection rules :	- Si elles existent et si elles conviennent, alors appliquer celles récupérées ; - Si elles n'existent pas, alors les créer.
Sub-Goal 2		- Nommer ces rubriques.
Operator		- Trouver un nom qui recouvre la fonction de la rubrique (ce qu'elle est sensée faire) et sa localisation dans le dictionnaire.
Methods		- Indiquer sur la feuille le nom de cette rubrique.
	Selection rules	- Si le nom de cette rubrique dépasse 8 caractères (limite autorisée pour la saisie), rejeter le nom. Sinon l'accepter.
Sub-Goal 3		- Donner un libellé à ces rubriques.
Operator :		- Choisir un libellé compréhensible pour les futurs utilisateurs.
Methods		- Indiquer sur la feuille le libellé de ces rubriques.
	Selection rules	- Les libellés doivent être proches du vocabulaire des utilisateurs (ils apparaissent généralement dans les spécifications générales fournies par les analystes).
Sub-Goal 4		- Donner une position de ces rubriques et de ces libellés à l'écran.
Operator :		- Indiquer si c'est une position absolue ou relative.
Methods		- Indiquer sur la feuille par un petit signe distinctif ("A" pour Absolu, "R" pour relatif).
	Selection rules	- Lorsqu'ils existent plusieurs rubriques sur un même écran, donner un positionnement absolu ; sinon relatif.
Sub-Goal 5		- Paramétrer l'aspect du libellé, des rubriques, des phrases de commentaire.
Operator :		- Indiquer si ces entités seront rouges ou blanches, en surbrillances ou en caractères normaux, en majuscules ou en minuscules...
Methods		- Indiquer sur la feuille, le paramétrage de ces rubriques (par des codes appropriés).
	Selection rules	- Si c'est un libellé, alors le paramétrer en majuscule ; - si c'est une rubrique, alors la mettre en caractère normal ; - si la phrase de commentaire est importante, alors utiliser le rouge ; - etc.

Sub-Goal 6	- Etablir l'enchaînement des écrans.
Operator :	- Indiquer l'ordre d'apparition des écrans : tel écran va succéder à tel autre.
Methods	- Décrire cette hiérarchie d'affichage par un diagramme sur le document de maquettage.
Selection rules	- Si c'est un écran de présentation de données, le placer après un écran de requêtes ; - si c'est un écran de présentation de résultats, le placer après un écran de traitements ; - Etc.

Figure 54

Formalisme appliqué à l'analyse organique (Experts Pacbase)
(lorsqu'il n'existe pas d'écrans préexistants à récupérer)

2. Une fois le plan de maquettage arrêté, l'informaticien se livre à la réalisation de ces sous-buts dans Pacbase

Goal :	- *Elaborer la maquette à partir du plan de conception défini durant l'analyse organique*
Va induire différents sous-buts :	
Sub-Goal 1'	- Appel de ces rubriques dans le dictionnaire.
Operator :	- Se placer sur la colonne correspondante de la grille de maquettage.
Methods	- Indiquer le code de la rubrique à récupérer.
Selection rules :	- Si première rubrique a été appelée, alors passer à la seconde. - S'il n'existe plus de rubriques à appeler, alors passer à leur dénomination.
Sub-Goal 2'	- Nommer ces rubriques à l'écran (par exemple, date de début de compte, de fin de compte).
Operator	- Se placer sur la colonne correspondante de la grille de maquettage par les touches du clavier.
Methods	- Indiquer le nom de cette rubrique.
Selection rules	- Si la première rubrique a été nommée, alors passer à la seconde. - S'il n'existe plus de rubriques à nommer, alors passer à la définition de leur libellé.
Sub-Goal 3'	- Fournir un libellé à ces rubriques
Operator :	- Se placer sur la colonne correspondante de la grille de saisie.
Methods	- Indiquer le nom de ce libellé.
Selection rules	- Si le premier libellé a été nommé, alors passer au second. - S'il n'existe plus de libellé à nommer, alors passer au positionnement des rubriques et des libellés.
Sub-Goal 4'	- Positionnement de ces rubriques et des libellés à l'écran.
Operator :	- Se placer sur la colonne correspondante de la grille de saisie.
Methods	- Indiquer leurs coordonnés et leur emplacement en absolu ou en relatif.
Selection rules :	- Si la première rubrique a été positionnée, alors passer à la seconde ; - Sinon, passer au positionnement de leur libellé.
Sub-Goal 5'	- Paramétrer les libellés, rubriques et commentaires.
Operator	- Se placer sur les colonnes correspondantes de la grille de maquettage.
Methods	- Indiquer les paramètres souhaités par les codes adéquates.
Selection rules	- Si la description de l'écran est terminée, passer à un autre écran. - Si la description de tous les écrans est terminée, passer à la définition de leur enchaînement.

Sub-Goal 6'		- Définition de l'enchaînement des écrans.
Operator		- Se placer sur la fonction appropriée du squelette de programmation .
	Methods	- Déterminer par une codification comment les écrans vont se succéder les uns à la suite des autres.
	Selection rules	- Si la définition de l'enchaînement des écrans est terminée, passer à leur simulation. - Si la simulation est correcte, alors passer à la programmation de l'application. - Sinon, corriger en conséquence.

Figure 55
Formalisme GOMS appliqué à l'activité de réalisation de la maquette
(Experts Pacbase)

Commentaire

La tâche apparaît fortement planifiée puisqu'on comptabilise 14 sous-buts qui vont se succéder pour la construction des écrans de Pacbase. On remarque aussi que la phase d'analyse organique est très importante dans la mesure où elle spécifie l'ordre de réalisation des actions de maquettage. Il y a d'ailleurs une symétrie parfaite entre les actions prévues en amont, lors de l'analyse organique, et celles qui sont mises en œuvre durant la composition des écrans.

Il convient à présent de confronter le déroulement effectif de l'activité à cette structure idéale de tâche de maquettage.

3.1.2.1.2 Une planification hiérarchisée de l'activité chez les experts Pacbase

Ces observations ont permis de scinder l'activité de maquettage en deux étapes complémentaires : 1) *l'analyse organique* et 2) *la réalisation de la maquette.*

1) Analyse organique

L'analyse organique représente la phase écrite des spécifications de l'application (tant au niveau des écrans à concevoir que du programme à implémenter) : en fonction du cahier des charges qui lui a été remis par l'analyste, le programmeur décrit méthodiquement une solution à implémenter dans le système, sous forme de pseudo-codes[52] et d'organigrammes :

> *" Au niveau de l'analyse organique, dans le pseudo-code je vais mettre une grosse fonction, par exemple "calcul = convertir un cours un francs" ou encore une phrase comme "convertir un cours en francs". Cela pourra être un algorithme relativement complexe derrière".*

[52] Pseudo-code : cela correspond à un mélange de codes informatiques et de commentaires en langage naturel rédigés sur un document de conception.

Ce plan de résolution suit fidèlement l'organisation des fonctions de base des différents canevas de conception (squelettes de programmation et grille de maquettage). Mais le fait le plus intéressant à noter est que l'informaticien s'engage déjà dans la consultation du dictionnaire pour identifier les entités informatiques (écrans, bouts de programmes...) qu'il pourra récupérer plus tard, lors du maquettage sur Pacbase. La réutilisation massive de composants techniques a pour vocation d'alléger la charge de travail et de diminuer les temps de développement. Cela étant, cette opération n'est possible que grâce au contexte relativement sclérosé et récurrent de la conception site-central. Nous y reviendrons plus tard.

« Il m'ait arrivé de créer de toutes pièces une maquette, mais je perds tellement de temps à essayer de me rappeler ce que j'ai mis là, que je reprends en général un modèle, quitte à le modifier structurellement. »

2) Réalisation

Une fois le plan clairement établi, l'informaticien débute véritablement l'étape de réalisation de la maquette. En fait, il s'insère dans les zones saisissables des canevas de conception : squelettes de programmation (TP ou Batch) et grille de maquettage. La programmation n'est plus alors que la transposition des spécifications écrites en instructions Pac. Le programmeur suit le plan de développement qu'il a établi au cours de l'analyse organique et tente de minimiser les digressions entre les spécifications prévues et le codage réalisé. Le modèle de conception s'assimile dès lors parfaitement à l'ossature des canevas de conception.

« Pacbase a quand même des contraintes et des fois, je le vis difficilement; on doit par exemple subir son ossature de traitement, et on est obligé de faire avec. On est obligé de se plier à la philosophie de Pacbase »

La moindre digression par rapport à ces prescriptions fonctionnelles conduit immédiatement au blocage du système. Ce dernier n'accepte en effet aucune autre démarche que celle qu'il a prévue.

« Il y a un déroulement logique qu'il vaut mieux respecter, parce que sinon Pacbase nous jette ! »

Si l'on compare ces observations à la représentation hiérarchique, il ressort que cette organisation correspond globalement à celle qui avait été décrite dans la structure de buts. Une légère différence existe pourtant au niveau de la phase organique. En effet,

les buts formulés par les informaticiens dans le cadre des observations de terrain nous ont paru non seulement moins nombreux, mais aussi plus généraux que ceux déterminés dans la structure de buts. Ce qui ne veut pas dire que la tâche soit moins planifiée. Simplement, les objectifs planifiés couvrent davantage de cibles et sont plus stéréotypés. En voici les raisons :

- Tout d'abord, les contraintes techniques limitent fortement toutes innovations de développement : un plan de maquettage reste souvent identique d'un projet à un autre. Aussi, lorsque l'informaticien détermine son plan, seuls les buts originaux ou complexes seront spécifiés (*comme par exemple, une rubrique qui sera créée de toute pièce et non récupérée*). Les autres buts, plus familiers au projet, ne feront l'objet d'aucune désignation formelle. Autrement dit, même si ces buts n'apparaissent pas explicitement sur le plan, ils existent tacitement et seront exécutés durant le maquettage. Ce qui revient à dire qu'une partie du plan de résolution est automatisée.

> *Par exemple :*
> le sous-but 5 intitulé "*Paramétrage des libellés, des commentaires et des rubriques*" de la structure de but (Cf.Figure 54) est un but standard à tous les développements. Les informaticiens expérimentés ne prennent donc plus la peine de le formuler explicitement dans leur plan de maquettage. Ils l'exécuteront directement dans le cadre de la réalisation de l'écran. Il en va de même pour le positionnement des rubriques.

- Une deuxième explication est que des buts peuvent être combinés avec d'autres buts. Ces agrégations permettent de faire des économies cognitives de traitement et de gagner du temps dans l'élaboration du plan de développement.

> Par exemple :
> On a observé que les deux sous buts : "*appeler la fiche de définition d'un écran* " et " *faire un diagnostic*" sont réunis en un seul but "*évaluer l'écran récupéré*". Ce type de conduite requiert cependant une très grande maîtrise de la conception.

- Une troisième raison est liée à la réutilisation d'anciens projets. En effet, l'informaticien ne sera plus obligé de définir des objectifs à atteindre s'ils sont déjà réalisés dans le projet récupéré. Le composant récupéré correspond donc à une sorte de méta-objectif qui englobe une multitude de sous-buts. La Figure 56 en présente d'ailleurs une illustration : nous avons reconstitué une représentation hiérarchique de l'analyse organique à partir des données de la verbalisation provoquée.

Par exemple :
les 6 sous-buts de la structure de buts originelle de l'analyse organique "*Définir un écran*", "*Définir une rubrique*", "*Définir un libellé*", "*Positionner les rubriques*", "*Paramétrer l'aspect des rubriques*" et "*Etablir la cinématique d'enchaînement des écrans*" (Cf.Figure 54) peuvent être remplacés par un objectif plus général : "*Désigner les écrans à récupérer dans le dictionnaire (écrans présentant les mêmes caractéristiques que l'application à développer*") (Cf. Figure 56)

Goal :	- Elaborer le plan d'action général de maquettage durant **l'analyse organique** lorsqu'il **existe des écrans à réutiliser**
Va induire ces sous but :	
Sub-Goal 1	- Désigner les écrans à récupérer dans le dictionnaire.
Operator :	- Vérifier qu'ils conviennent et les désigner sur le document de maquettage..
Methods	- Fonction de recherche d'écran dans le dictionnaire.
Selection rules :	- Si ils conviennent parfaitement, alors réutiliser ces écrans en l'état. - Si ils ne correspondent pas totalement aux spécifications détaillées, alors les modifier en conséquence.
Sub-Goal 2	- Définir l'enchaînement des écrans récupérés.
Operator :	- Indiquer l'ordre d'apparition des écrans : tel écran va succéder à tel autre.
Methods	- Décrire cet hiérarchie d'affichage par un diagramme sur le document de maquettage.
Selection rules	- Si c'est un écran de présentation des donnée, le placer après un écran de requêtes ; - si écran de présentation des résultats, le placer après un écran de traitements ; - Etc.

Figure 56
Formalisme GOMS appliqué à l'analyse organique (Experts Pacbase)
(lorsque l'informaticien peut récupérer des écrans préexistants)

– Un dernier argument repose sur l'expérience que les informaticiens ont acquise dans l'environnement Pacbase : à force de développer des plans qui s'accordent parfaitement bien avec les canevas de programmation, l'informaticien a fini par accommoder son fonctionnement mental –ici, son modèle d'action– au paradigme de ces dispositifs. Cette "prédisposition cognitive" lui permettrait dès lors de proposer directement des plans qui s'ajustent parfaitement bien à l'organisation des squelettes. Cette dernière remarque est importante car elle implique que tout informaticien qui migre vers le client serveur est porteur d'un principe de fonctionnement mental étroitement associé au modèle procédural de Pacbase.

Pour résumer, par rapport à la structure de buts exposée initialement, on retrouve la même partition générale de l'activité entre "l'analyse hors écran" et "la réalisation sur écran". Mais le fait le plus remarquable concerne le rôle que joue les structures techniques dans la planification du maquettage. L'informaticien organise ses actions moins en fonction de ce qu'il souhaite faire, que de ce que le modèle récupéré et le système technique lui permettent de faire.

3.1.2.2 La planification dans NSDK

3.1.2.2.1 Structure de buts définie pour la tâche de maquettage dans NSDK

Sur la base des entretiens exploratoires effectués, la tâche de maquettage a pu être décomposée hiérarchiquement en trois étapes complémentaires :

1. d'abord, une représentation papier de la maquette à concevoir
2. ensuite, la réalisation graphique de cette maquette avec l'éditeur graphique de NSDK
3. enfin, la programmation des événements de la maquette.

1. **Représentation physique** de la maquette sur une feuille

Goal :	- ***Elaborer une représentation papier des fenêtres à concevoir***
Va induire différents sous-buts :	
Sub-Goal 1	- Représenter une première fenêtre.
Operator :	- Choisir entre différentes représentations possibles (fenêtre générale de travail, fenêtre boîte de dialogue, fenêtre de commentaire...).
Methods	- Dessiner la fenêtre à main levée.
Selection rules :	- Si c'est une fenêtre générale (appelée aussi fenêtre mère), alors la dessiner de manière plus imposante que les autreS fenêtres.
Sub-Goal 2	- Nommer la fenêtre.
Operator	- Trouver un nom qui recouvre la fonction de la fenêtre (ce qu'elle est sensée faire) .
Methods	- Indiquer le nom de la fenêtre sur sa barre de titre.
Selection rules	- Si le nom de cette fenêtre n'a pas été jugé suffisamment explicite par l'ergonome, alors rejeter le nom. Sinon l'accepter.
Sub-Goal 3	- Représenter les objets sur cette fenêtre.
Operator :	- Choisir entre différentes représentations possibles (boutons, ascenseurs, menus...).
Methods	- Dessiner rapidement les boutons à main levée.
Selection rules :	- Si ce sont des boutons de fermeture d'une fenêtre, de validation ou d'annulation d'une action, alors la placer en bas à droite de l'écran. - Si ce sont des menus, alors les placer en haut de l'écran ; etc.

Sub-Goal 4		- Nommer ces objets graphiques.
Operator :		- Trouver un nom qui recouvre la fonction de ces objets et qui correspondent au vocabulaire du métier .
Methods		- Indiquer le nom des objets sur la feuille.
	Selection rules	- Si les noms donnés à ces objets n'ont pas été jugés suffisamment explicites par l'ergonome), les rejeter. Sinon les accepter.
Sub-Goal 5		- Description des événements qui vont se déclencher suite aux actions des utilisateurS.
Operator :		- Définir les événements susceptibles de se dérouler.
Methods		- Indiquer par une flèche l'événement que peut engendrer un objet sélectionné sur l'interface (calcul, grisage de certaines options du menu, ouverture d'une fenêtre...).
	Selection rules	- Deux événements devront être cohérents ensemble (par exemple fermeture d'une fenêtre et ouverture d'une autre), etc.
Sub-Goal 6		- Description des fenêtreS qui sont susceptibles de s'ouvrir suite aux actions des utilisateurs.
Operator :		- Faire la distinction entre les fenêtres filles, les boîtes de dialogue, les fenêtre de messages.
Methods		- Désigner par une flèche les fenêtres qui vont s'ouvrir suite aux actions de l'utilisateur.
	Selection rules	- Si c'est une fenêtre de message (commentaire, message d'erreur, validation), alors la laisser en multifenêtrage. - Si c'est une fenêtre fille alors faire disparaître la fenêtre mère, etc.

Figure 57

Structure de buts de la description physique de l'interface (sur NSDK)

2. Transposition du dessin sur l'écran : étape de **réalisation** graphique de la maquette.

Goal :		*Réaliser l'interface à l'aide des outils de NSDK : Conception graphique de la maquette*
Va induire différents sous-buts :		
Sub-Goal 1		- Sélectionner une première fenêtre.
Operator :		- Utiliser la souris et la manipulation directe.
Methods		- Amener la fenêtre souhaitée de la boîte à outils au plan de travail par la manipulation directe, puis la tailler à la dimension souhaitée avec la souris (selon les mesures définis dans lacharte graphique).
	Selection rules	- Une fois que le format de la fenêtre a été établi, lui donner un nom.
Sub-Goal 2		- Nommer la fenêtre.
Operator		- Utiliser les boutons de la souris.
Methods		- Sélectionner la fenêtre avec le bouton gauche de la souris, puis appuyer sur le bouton droit pour afficher sa boîte dialogue correspondante. Y indiquer le nom de la fenêtre qui apparaîtra dans sa barre titre.
	Selection rules	- Si le nom de la fenêtre est indiquée, alors passer à la sélection et à la disposition des objets graphiques.
Sub-Goal 3		- Sélection et disposition des objets sur cette fenêtre.
Operator :		- Utiliser la souris par manipulation directe.
Methods		- Amener les objets souhaités de la boîte à outils au plan de travail par manipulation directe, puis les disposer sur le plan de travail, en tenant compte des espaces et des dimensions définis dans la charte graphique.

	Selection rules :	- Si ce sont des boutonS de fermeture d'une fenêtre, de validation ou d'annulation d'une action, alors la placer en bas à droite de l'écran, à des distances de X pixels du cadre et de Y pixels de son plus proche voisin. - Si la disposition des boutons est terminée, alors passer à leurs dénominations.
Sub-Goal 4		- Nommer les objets graphiques.
Operator :		- Utiliser les boutons de la souris.
Methods		- Sélectionner l'objet graphique avec le bouton gauche de la souris, puis appuyer sur le bouton droit pour afficher sa boîte dialogue correspondante. Y indiquer le nom de l'objet
	Selection rules	- Une fois le nom de cet objet définie passer à sa programmation.

Figure 58

Structure de buts de la réalisation graphique de la maquette (sur NSDK)

3. Programmation des composants de la maquette

Sub-Goal 5	- Choix des événements	
Operator :		- Utiliser les boutons de la souris et sélectionner un ou plusieurs événements parmi une liste prédéfinie.
Methods		- Sélectionner l'objet graphique avec le bouton gauche de la souris, puis appuyer sur le bouton droit pour afficher sa boîte de dialogue correspondante. Sélectionner avec la souris le ou les événements souhaité (s).
	Selection rules	- Lorsque les événements sont sélectionnés, passer à leur programmation.
Sub-Goal 6		- Programmation des événements.
Operator :		- Utiliser la souris, les librairies de fonctions et le clavier pour la codification.
Methods		- Se placer avec la souris sur une zone "Programme" de la boîte de dialogue où apparaissent les événements sélectionnés. Spécifier le comportement particulier de chaque événement en le programmant et en intégrant des fonctions préfabriquées le cas échéant.
	Selection rules	- Lorsque la programmation des événements est terminée, passer à d'autres objetS. - Si la programmation de tous les objets est terminée, passée à leur simulation ou à la conception d'une autre fenêtre.
Sub-Goal 7		- Simulation du bouton, de la fenêtre ou de l'interface.
Operator :		- Utiliser les fonctions de simulation de NSDK .
Methods		- Sélectionner l'icône graphique symbolisant la simulation.
	Selection rules	- Si le test de l'interface donne les résultats attendus, alors se livrer au regroupement du code dans une librairie unique. - Sinon, modifier le programme ou ajouter de nouvelles fonctions sur la maquette.
Sub-Goal 8		- Regroupement du code dans une seule librairie.
Operator :		- Sélectionner le code avec la souris et le rassembler dans une même zone.
Methods		- Se placer avec la souris sur la zone "Programme". Sélectionner le code, utiliser la fonctions "Copier". Puis ouvrir la librairie où est stockée le programme générale la fenêtre. "Coller" le code.
	Selection rules	- Lorsque la concentration de ce code est terminée, alors passer à la conception d'une autre fenêtre. - Si la conception de toutes les fenêtres est terminée, alors effectuer une simulation générale en testant différents scénarios et en corrigeant les incohérences observées le cas échéant.

Figure 59

Structure de buts de la programmation de la maquette (Experts NSDK)

On notera tout d'abord le nombre très important de sous-but planifiés ; près d'une quinzaine. Si on compare cette structure de buts avec celle de Pacbase (10 sous-buts), on serait tenter de dire qu'elle est plus hiérarchisée puisqu'il y a plus de buts prédéfinis. Seulement, il faut prendre en compte l'étape de programmation qui est une phase nouvelle dans la conception client serveur, et qui n'existait donc pas en site-central. Ces nouveaux buts correspondent à des choix d'événements et de fonctions ainsi qu'à des actions de programmation sur la maquette.

3.1.2.2.2 Une planification située chez les experts NSDK

Les observations effectuées auprès des experts NSDK ne donnent pas une partition aussi formelle du processus de maquettage que dans la structure de buts. Au contraire, ces analyses révèlent plutôt une intrication étroite entre les différentes phases de la conception.

Ces experts se dispensent de la phase préalable "papier" pour se lancer dans la réalisation de la maquette. Ils exécutent directement sur écran leur plan de conception. C'est pratiquement une constance pour des compositions simples et classiques, comme une fenêtre avec un message d'erreur. Pour des réalisations plus perfectionnées, d'aucuns préfèrent griffonner sur un bout de papier, un croquis très grossier indiquant, par exemple, les relations logiques (ouverture/fermeture) entre différentes fenêtres mères et filles. L'informaticien y décline alors des objectifs très généraux, qui recouvrent, de manière implicite, de multiples sous-buts non déclarés :

> *Par exemple :*
> Le but "*composer une fenêtre commentaire*" est le composite d'un ensemble de sous-buts comprenant les sous-buts "*Appeler une fenêtre dialogue*", "*Composer un texte*", "*Placer un bouton Ok*", "*Lier cette fenêtre à un événement d'action irréversible*".
> L'informaticien ne prend pas la peine de détailler cette procédure sur la feuille ou verbalement, mais désigne simplement l'objectif général : "*composer une fenêtre commentaire*"

Cela dit, dans la plupart des cas étudiés, les informaticiens construise leur plan de maquettage au fur et à mesure qu'il réalise les fenêtres. La simulation leur permet d'évaluer, en temps réel, le coût (temps de réponse) et la pertinence (convivialité) de ces buts de conception. Cette construction "contextuelle" du plan dans le cours de la conception, n'est possible que grâce à la souplesse de l'environnement qui peut accepter plusieurs versions de solution de maquettage pour un même projet. Nous en

verrons d'ailleurs des illustrations dans le paragraphe abordant les possibilités techniques de changer le plan.

> *« On peut tester chaque fenêtre en temps réel. On n'est donc pas obliger de tout compiler, on peut le faire au fur et à mesure. Donc on essaie en fait. C'est du tâtonnement, c'est de l'ajustement un petit peu à chaque fois »*

Ici, c'est donc surtout l'écran qui sert de support à l'informaticien pour planifier un schéma de résolution, et moins la feuille. On est dans le cas d'une "*planification située*" de la solution qui se construit dans l'action, et non pas en suivant un plan préalablement défini. Analysons maintenant la conduite des novices.

3.1.2.2.3 Une planification hiérarchisée puis opportuniste de l'activité chez les novices NSDK

L'organisation de l'activité observée chez le novice relève davantage de la structure de buts que du comportement des experts NSDK. On trouve en effet une séparation assez nette entre l'étape de préparation/planification de l'activité d'une part, et de conception/programmation de la maquette d'autre part.

a) Organisation planifiée de l'activité dans la description physique de la fenêtre

Cette étape consiste à décrire très précisément l'aspect et le comportement des futures fenêtres sur une feuille[53]. L'informaticien représente ainsi tous les écrans qu'il souhaite concevoir et agrémente ces croquis de commentaires très détaillés (noms des objets, des événements, des fonctions appelées...). Cette description peut être considérée comme un plan de conception dans la mesure où les objets qui y sont spécifiés correspondent à autant de buts de maquettage qui baliseront les actions futures du concepteur.

Par exemple : dans cette étape de description physique de la maquette, on trouve un grand nombre d'indications (ou sous-buts) relatifs à la définition d'un bouton graphique. Le novice y désigne le type de bouton à afficher, sa couleur, sa dénomination, son emplacement, son événement, sa fonction, son programme ainsi que son comportement et les actions que ce contrôle est sensé déclencher sur les autres composants de l'interface (ouverture d'une fenêtre, fermeture...). Cette description se révèle beaucoup plus riche que celle de certains experts NSDK qui, pour la même tâche, se contentait de griffonner sommairement un esquisse de fenêtre ; les plus aguerris s'engageant même directement dans la composition avec NSDK.

[53] Un exemplaire est présenté dans l'Annexe VII

Aussi, plus le débutant va affiner ses intentions de conception sur la feuille, plus il disposera d'un guide de développement précis et stable.

> *« Dans la mesure où il y a un nombre d'outils plus important, on se pose beaucoup plus de questions : comment faut-il organiser les objets ? Il faut se créer un projet, il faut déterminer les choses dont on a besoin sinon cela ne marche pas. Et cela n'est pas du tout évident. On n'avait pas à se soucier de ça avec Pac, on n'avait pas de questions à se poser ! »*

Comme on peut le constater dans cet extrait de corpus, les sous-buts remplissent également une autre fonction ; celle de rassurer le développeur débutant. L'absence de guides de conception –semblables aux canevas de Pacbase– le déstabilise en effet. C'est pourquoi la planification méthodique d'une solution avec un très grand nombre de buts, pourra être vue comme le moyen de réduire l'incertitude inhérente à cette nouvelle tâche.

A des fins de comparaison, nous avons tenté de représenter la phase de description papier de la maquette selon le même principe que sa structure de buts (Figure 60). Malgré la similitude entre ces deux configurations, on comptabilise un plus grand nombre de buts chez les novices (9 sous-buts) que dans sa structure de buts de référence (6 sous-buts). Cette majoration s'expliquerait donc par le recours à un nombre plus important de repères pour l'action.

Goal :	- *Fournir une description générale des fenêtres à concevoir sur du papier*
Va induire différents sous-buts :	
*Sub-Goal 1**	- Représenter une première fenêtre.
*Sub-Goal 2**	- Nommer la fenêtre.
*Sub-Goal 3**	- Représenter les objets sur cette fenêtre.
*Sub-Goal 4**	- Nommer ces objets graphiques.
**Identiques à ceux de la structure de buts*	
Sub-Goal 5 — *Nouveaux buts*	- Description des actions susceptibles de se réaliser suite aux interventions des utilisateurS.
Operator :	- Définir les actions sous forme d'opérateurs génériques (fermeture d'une fenêtre, ouverture de telle autre, calcul de telle fonction, changement d'état d'un icône ou d'un menu...).
Methods	- Indiquer ces actions vers les boutons qui sont sensés les déclencher.
Selection rules	- Conserver une cohérence entre ces actions.

*Sub-Goal 6**		- Description des événements qui vont se déclencher suite aux actions des utilisateurS.
Sub-Goal 7		- Description succincte des fonctions de programmation des événements.
Operator :		- Identifier les fonctions qui peuvent être réutilisées pour la programmation .
Methods		- Nommer ces fonctions et indiquer leur localisation dans les librairies.
	Selection rules	- Si les fonctions sont identiques à celles recherchées, alors les réutiliser.
		- Sinon, les créer.
*Sub-Goal 8**		- Description des fenêtreS qui sont susceptibles de s'ouvrir suite aux actions des utilisateurs.
Sub-Goal 9		- Description générale des liens entre les fenêtres de l'interface.
Operator :		- Identifier les types de relations entre les fenêtre (fenêtre mère, fenêtre fille, fenêtre qui s'affichera simultanément ou successivement à une autre).
Methods		- Indiquer ces relations sous forme d'un organigramme.
	Selection rules	- Ne pas surcharger l'écran en fenêtres, sinon diminuer le nombre de fenêtre, etc.

Figure 60
Formalisme GOMS appliqué à l'activité de spécification de l'interface (novices NSDK)

Après avoir terminé cette représentation papier, l'informaticien est prêt pour se livrer à la réalisation sur NSDK. Il s'inspire de ce "canevas" pour dérouler les étapes de la conception.

b) Remise en question du plan durant la réalisation

Les informaticiens novices effectuent de nombreuses déviations par rapport au plan annoncé. Etant peu expérimentés dans la programmation événementielle, ces débutants ont planifié des objectifs de maquettage relativement confus, dont l'efficacité est éprouvée par les nombreuses simulations effectuées tout au long de la réalisation (plus d'une vingtaine en moyenne).

Par exemple :
On note que ces tests dévoilent souvent des incohérences entre les événements de la maquette. Dans la plupart des cas, l'informaticien corrige alors dynamiquement son plan en redéfinissant de nouveaux sous-buts à atteindre, en particulier ceux relatifs au "*choix des événements*" (Sub-goal 5), à leur "*programmation*" (Sub-goal 6) et à la "*sélection des fonctions de programmation*" (Sub-goal 7) (Cf Figure 60).

Ces tests négatifs peuvent être à l'origine d'une remise en cause partielle (évolution de certains buts) ou totale (redéfinition complète du projet) du plan de maquettage.

> C'est par exemple le cas lorsque l'informaticien s'est fixé comme objectif de maquetter une fenêtre supportant de multiples dispositifs de commandes (boutons, icônes, menus, listes déroulantes...) qui, non seulement, rendent plus difficile sa manipulation, mais de surcroît, accentuent les risques de défaillance (incohérence entre les nombreux événements). Devant les mauvaises performances de sa maquette, l'information n'a pas d'autres choix que de redessiner un nouveau de plan de maquettage, et de redécomposer la fenêtre générale en plusieurs sous-fenêtres filles, associées à des boîtes de dialogue plus spécifiques

En définitive, la gestion de cette activité de maquettage se fait d'abord selon un mode hiérarchisé, puis opportuniste. Seulement, à la différence des experts, les déviations et les corrections se font toujours par rapport à un guide préétabli ; alors que chez les programmeurs chevronnés, la construction du plan s'effectuait dans le cours même de l'action.

3.1.2.2.4 En résumé

Par rapport à la structure de buts de Pacbase qui comportait deux phases (*analyse organique + paramétrage des écrans sur grille de saisie*), celle de NSDK comporte une étape de plus (*description physique des fenêtres + conception graphique par manipulation directe + programmation des événements de la fenêtre*). On compte d'ailleurs 10 sous-buts dans la première structure, contre 14 sous buts dans la seconde.

Cela dit, ces structures sont idéales, c'est-à-dire qu'elles représentent ce qui devrait être normalement fait et non ce qui effectivement réalisé. C'est l'observation de l'activité qui nous a permis de le déterminer.
On a ainsi montré que les experts Pacbase arrivait à diminuer le nombre de sous-buts (-4) par la définition d'un objectif plus général de récupération d'écrans préexistants. On peut donc dire que la structure de buts surestimait d'une certaine manière le nombre de buts réellement planifié dans l'activité.

De la même manière, la structure de buts de la tâche en client serveur suggérait un nombre de buts plus élevé que ce qui a pu être constaté au cours de l'analyse de l'activité des experts NSDK. En fait, la composition d'un scénario de maquettage est plus souvent réalisé directement sur l'écran que sur le papier. Aussi, l'expert ne décrit pas cette étape (et par là même, ses sous-buts afférents) comme une phase à part de la

réalisation graphique, mais comme intégrée à celle-ci. C'est d'ailleurs pour cette raison que l'on se trouve dans le cadre d'une planification située puisque l'informaticien organise circonstanciellement le déroulement de la conception en fonction des résultats des tests entrepris sur ses idées de maquettage.

A l'inverse, la structure de buts s'est révélée moins exhaustive que l'activité des novices NSDK. On a en effet identifié des buts que la représentation hiérarchisée n'avait pas pris en compte, en particulier ceux émis durant la phase de description des fenêtres sur le papier. Là où le modèle hiérarchique en déclinait 6, le novice en donnait 9. Cette augmentation peut s'entendre comme le besoin pour l'informaticien de jalonner plus profondément son activité, afin de supprimer toute indétermination dans le maquettage.

C'est au cours de l'exécution du plan que le novice prendra conscience qu'il ne peut pas se tenir aux objectifs fixés, soit parce qu'ils sont conceptuellement trop difficiles à réaliser par rapport à son niveau expérience, soit parce qu'ils sont trop coûteux, ou bien encore, parce qu'ils sont bien moins fiables que prévus. Il devra alors se livrer à des réorganisations sur le plan selon une démarche opportuniste ; voire dans des cas plus extrêmes, à revenir sur la totalité de son projet de maquettage.

On se trouve donc dans une situation où la planification de l'activité s'impose d'abord, mais où intervient ensuite, une réorganisation opportuniste de celle-ci en fonction des résultats obtenus.

Cela dit, ce balisage méticuleux du maquettage peut très vite devenir un carcan pour l'informaticien. Il cherchera en effet à tout prix à respecter le plan de conception, au risque de manifester une trop grande rigidité pour gérer les aléas de la conception. Dans les cas extrêmes, il se refusera même à effectuer des réajustements, mêmes ponctuels, susceptibles de compromettre l'application de son scénario. Ce n'est que lorsqu'il ne pourra vraiment plus avancer qu'il sera finalement amener à revoir toute la procédure et à redéfinir un nouveau scénario de maquettage.

Nous allons à présent déterminer dans quelle mesure les systèmes techniques interviennent dans la construction et la gestion de ce plan de maquettage.

3.1.2.3 Les capacités à changer de plan

Dans le cadre de l'activité, il arrive souvent qu'une erreur ou qu'une nouvelle idée de conception remette en cause le plan qui avait été prévu initialement. Au cours de nos observations, nous nous sommes intéressés à la façon dont l'environnement technique permettait de modifier et/ou d'adapter opportunément le plan d'action à de nouvelles orientations de maquettage.

a) Limitation des possibilité de changement de plan sur Pacbase

On peut d'ores et déjà dire que les nombreuses contraintes techniques de Pacbase empêchent tout changement de plan de maquettage. Ses canevas de conception n'admettent en effet pas que l'informaticien développe autre chose que ce qui a été prévue pour eux. Comme nous le laissions déjà entendre auparavant, toute écart par rapport aux règles de conception se solde par un blocage immédiat du système. En outre, la simulation d'un écran affiche une image statique qui ne peut être animé. Dans ces conditions, les moyens techniques proposés se révèlent peu appropriés pour des explorations et des tests de développement. Enfin, chaque compilation génère un coût d'utilisation du site-central (CPU). L'informaticien restreint alors volontairement les tests, quand bien même ils pourraient lui permettre d'affiner son plan de maquettage ou d'expérimenter de nouvelles solutions.

b) Larges possibilités de changement de plan avec NSDK

NSDK se démarque fondamentalement de Pacbase par l'absence de structures administratrices de l'activité de conception. L'informaticien détermine donc seul ses interactions avec l'environnement technique. Il peut organiser et remanier son plan très librement, sans craindre les réactions du système. A titre d'exemple, le développeur peut construire une "*fenêtre fille*" avant "*une fenêtre mère*" ou bien simuler une seule fenêtre ou la totalité de la maquette. Ses marges de manoeuvre dans la conception sont donc très importantes.

> *« On a quand même un peu plus de liberté au niveau des outils, je pense à la compilation : on peut compiler toutes les 3 secondes. Si on a envie de déplacer un bouton, d'aller le mettre ailleurs, d'aller voir le code, on se ballade sur les fenêtres... c'est quand même plus agréable que sur site central où on est bloqué sur un écran : où on ne peut pas faire de copier/coller. Il y a donc une liberté pour le programmeur qui est tout autre. »*

Cette fonction de simulation favorise la remise en cause du plan de conception car l'informaticien peut expérimenter à tout moment ses idées de conception et, selon les résultats obtenus, réajuster son projet de maquettage.
Enfin, la maintenance des versions de programmes ("*Versionning*") permet un style de conception très exploratoire : le programmeur ose tester différents agencements ou comportements de l'interface, les comparer et choisir les meilleurs. Il a toujours la possibilité de revenir à des états antérieurs de son programme puisque chaque version est conservée. De la sorte, le concepteur peut trouver un solution qu'il pourra décider d'intégrer opportunément à son programme de maquettage, à supposer qu'elle se révèle plus efficace que celle à laquelle il avait pensé initialement.

En définitive, il apparaît que le facteur technique joue un rôle très important dans la régulation de l'activité. Dans certains cas, le système se révèle assez souple et ouvert pour permettre une réorganisation complète ou un réajustement ponctuel du plan de conception (cas de NSDK). Dans d'autres cas, au contraire, le système figé contraint l'informaticien à bien préparer le plan adéquat et à le suivre, au risque de procéder à de complexes et coûteuses adaptations.

3.1.3 Synthèse - Discussion

Dans l'ensemble, ces observations sont compatibles avec nos prévisions puisque nous pronostiquions une organisation de l'activité différente selon le contexte technique dans lequel elle se déroulait : une planification hiérarchisée tout au long du maquettage avec Pacbase, une planification opportuniste avec NSDK. Cela dit, dans ce dernier cas, cette régulation peut se faire dans le cours de l'action (planification située des experts) ou par rapport à un plan prédéfini (planification opportuniste des novices)

a) L'organisation de l'activité chez les experts Pacbase : une planification hiérarchique

Les ossatures techniques de Pacbase imposent en effet à l'informaticien de mettre en relation son mode opératoire avec le déroulement des fonctions de programmation : il doit exprimer son plan de résolution sous forme de structures hiérarchisées qui correspondent à l'emboîtement des fonctions de base des canevas de Pacbase. Sans cette représentation hiérarchisée, son plan de conception risque d'être incompatible avec celui requis par l'environnement technique. En outre, ce plan est conservé et appliqué fidèlement durant toute la phase de réalisation. Sa remise en cause est difficile, tant les possibilités techniques (simulations peu performantes, canevas rigides) et les contrôles exercés par la hiérarchie (sur les dépenses CPU) modèrent toutes initiatives de conception.

En somme, le caractère éminemment prescriptif et parfois "répressif" de l'environnement laisse peut de place à des digressions opportunistes de conception. L'informaticien navigue dans un champ de possibilités dont les contours sont fixés arbitrairement par le système. Finalement, ses actions seront toujours définies en fonction des règles de l'environnement technique.

b) L'organisation de l'activité chez les experts NSDK : une planification située

NSDK se démarque de Pacbase par l'absence d'un aménagement imposé de l'activité de conception. L'informaticien est libre de s'organiser comme il l'entend. Il est l'artisan de sa propre démarche de travail.

Si un plan de maquettage existe, celui-ci est plutôt élaboré dans le cours de l'activité de maquettage. Sa mise en œuvre repose davantage sur les circonstances de la conception que sur des prescriptions techniques émanantes du dispositif. En d'autres

termes, l'activité se déroule plus selon un mode erratique au gré des circonstances et des découvertes de la conception que sur un mode hiérarchique, associé aux contraintes de l'outil. D'ailleurs, les nombreuses simulations réalisées conduisent à un ajustement contextuel du plan. L'informaticien peut ainsi expérimenter directement ses idées de conception, et aménager dynamiquement son projet de maquettage à partir des résultats obtenus.

Cette gestion contextuelle de l'activité, qui s'oppose à la planification hiérarchisée du site-central, nous amène à nous interroger sur la conduite que va tenir le novice face aux nouvelles contingences de conception proposées par le client serveur.

c) L'organisation de l'activité chez le novice NSDK : une planification opportuniste

En passant du site-central au client-serveur, l'informaticien se trouve donc affranchi du carcan du site-central. Il doit alors se séparer d'un modèle très réglementé de la tâche pour adopter un mode de gestion plus opportuniste. Conceptuellement, cela signifie qu'il doit faire abstraction de la plupart de ses automatismes de planification pour mettre en place des conduites de travail plus souples et réactives. En d'autres termes, il ne s'agit plus pour lui d'ajuster son plan d'action aux prescriptions du système, mais de l'adapter circonstanciellement au progrès de la résolution.

Ce changement technique constitue, de fait, un passage très difficile à assumer pour ces novices car, culturellement, cela implique de rompre avec des règles de maquettage qui restent, malgré tout, profondément enracinées en eux. Pour tous ces développeurs, Pacbase représente en effet l'unique référence en terme d'outils de programmation. Pendant de nombreuses années, ils ont appris à programmer, à concevoir des maquettes, à planifier des actions... en respectant les directives de cet environnement.

Mais, confrontés à l'utilisation de NSDK, ces débutants se trouvent, pour la première fois de leur carrière, livrés à eux-mêmes. Il faut qu'ils fixent et organisent seuls les étapes de la conception, qu'ils décident de manière totalement autonome des différentes actions à entreprendre, et enfin, qu'ils prennent l'initiative de sélectionner les ressources de l'environnement à utiliser. Ils ne peuvent plus compter sur les canevas techniques pour guider leur activité, ni sur les modèles préexistants pour les aider à planifier des solutions. Du coup, cette liberté d'action qui aurait pu paraître, à

bien des égards, comme un acquis fondamental de la transition technologique, se révèle paradoxalement problématique pour ces novices. L'absence brutale de repères le déstabilise plus qu'elle ne le rassure. Ce malaise se manifeste d'ailleurs au travers de la gestion qu'ils font des ressources techniques de l'environnement durant le maquettage. Deux conduites extrêmes se démarquent :

1. D'un coté, il y a ceux qui s'autocensurent volontairement en n'exploitant que quelques possibilités techniques de l'environnement. Les solutions proposées sont alors relativement pauvres et peu originales. En fait, ils prennent très peu de risques pour éviter de commettre des erreurs de conception.

   « J'ai l'impression que l'on est tellement obligé de se limiter, pour ne pas que cela dérive trop que cela en revient à des écrans comme ça [site-central] ».

2. De l'autre, il y a les informaticiens qui se livrent à une surexploitation anarchique et superfétatoire des possibilités de l'outil. C'est alors l'inverse qui se produit. L'informaticien teste toutes les ressources de l'environnement et planifie une solution qui se révèle finalement assez excentrique.

   « Le plus difficile, je trouve que c'est le maquettage : arriver à des écrans qui ne soient pas des arbres de Noël, qui ne clignotent pas de partout. »

Aussi, si quelques-uns furent tentés de profiter de ces nouveaux espaces de liberté, la grande majorité appréhende avec circonspection ces nouvelles voies de développement, par crainte de se perdre dans les méandres du développement graphique et événementiel.

Finalement, il ressort de cette première analyse que l'informaticien novice éprouve de réelles difficultés pour s'émanciper techniquement, c'est-à-dire pour s'affranchir des guides de conception. Il recherche la tutelle technique de NSDK alors que celui-ci se garde justement d'imposer des trames pour l'action. Il cherche des repères dans un environnement qui n'offre que des contingences d'action. Il n'en sera que plus désorienté :

« Avant, on avait le confort de la certitude. C'était sécurisant. Ce repère qu'on avait a disparu. Il est difficile de ne plus se référer à un signal aussi fort ».

3.2 MODÈLE DE RÉSOLUTION

3.2.1 Méthode employée

a) Rappel de l'hypothèse

Les processus de résolution de problème, qui concernent l'élaboration ou la récupération de solutions, tendent à évoluer de l'analogique vers le dynamique. L'informaticien n'ayant plus les moyens techniques de transposer directement les solutions récupérées sur un problème, il construira par ajustements progressifs la solution finale. En d'autres termes, il s'agit moins d'exécuter des procédures préalablement définies que de découvrir des solutions à des problèmes originaux. C'est pourquoi, de nouvelles stratégies cognitives devraient être également requises pour gérer les spécificités de la conception client serveur.

b) Techniques de recueil de données

Les techniques de recueil de données utilisées pour cette étude concernaient :

- D'une part, des *observations effectuées sur l'activité de maquettage* où différentes caractéristiques ont été retenues, en particulier : les types de données utilisées pour élaborer les solutions, les caractéristiques des problèmes rencontrés, les différents types d'erreurs commises par les informaticiens et les procédures de récupération déployées, les niveaux de fiabilité des solutions (nous les préciserons dans le prochain paragraphe). L'évaluation de ce dernier point s'est faite en collaboration avec les informaticiens experts et les prestataires qui accompagnaient les novices dans l'utilisation du nouveau système. Dès qu'une solution de maquettage ne menait visiblement pas au résultat attendu (manifestations bruyantes, remarques de l'information, interventions des collègues ...), nous leur demandions d'en expliquer les raisons.

  Ces observations ont porté sur la réalisation effective de la maquette et ont impliqué 5 informaticiens de chaque catégorie (experts Pacbase et NSDK, et novices NSDK). Elles ont duré 60 minutes environ.

- D'autre part, la *verbalisation provoquée.* qui a déjà fait l'objet d'une présentation dans le modèle d'action. Dans le cadre plus spécifique de cette analyse, nous nous sommes intéressés à la nature des solutions mises en oeuvre, à leur mode de construction, d'évaluation et de correction, à la gestion des problèmes rencontrés. Ces recueils ont été effectués lors des observations et duraient une trentaine de minutes en moyenne. On retrouve ici les mêmes informaticiens que ceux précédemment cités.

Etait considérée comme solution, toute procédure ou toute action qui était explicitement formulée et exécutée par l'informaticien pour atteindre un but ou résoudre un problème. Ce sont des actions élémentaires qui, une fois mises bout à bout, forment une séquence de résolution. Ces solutions se rapportent en fait aux "*Opérateurs*" qui avaient déjà été identifiés dans les structures de buts de la tâche (Cf. Modèle de l'action).

Exemple de solution exprimée : *« là, par exemple, j'ai mis ce que j'avais à mettre dans la fenêtre "Info". Ensuite, je vais sur les événements* (Solution 1) , *et je choisis celui qui m'intéresse* (Solution 2). *Par exemple là, sur "Selected" cela m'intéresse : quand je "Select", je fais quelque chose »*

Dans ce cas, par exemple, l'informaticien décrit la procédure pour sélectionner un événement en fonction d'un comportement attendu (but) : d'abord, (*i*) ouvrir la boîte de dialogue des événements ; puis, (*ii*) sélectionner l'événement idoine. Nous comptabiliserons donc deux solutions de programmation.

c) Indicateurs retenus et techniques d'analyse

Afin d'identifier les différentes stratégies de résolution mises en œuvre par chaque catégorie d'informaticien et, pour pouvoir les comparer entre elles, nous avons déterminé deux types d'indicateurs : l'un quantitatif, l'autre qualitatif.

1. En ce qui concerne les aspects **quantitatifs** de la résolution, nous avons comptabilisé le nombre moyen de solutions exprimées et exécutées par chaque informaticien, durant les phases de verbalisation provoquée. Et parmi ces solutions, nous[54] avons veillé à distinguer :

[54] Nous rappelons que l'évaluation de ces solutions était réalisée soit par l'informaticien lui-même, soit par le prestataire qui accompagnait le novice dans sa démarche de maquettage.

- Le nombre de solutions *incomplètes*, c'est-à-dire celles auxquelles il manque des éléments pour parvenir à une solution acceptable ;
- Le nombre de solutions *inappropriées*, c'est-à-dire celles qui ne sont pas appropriées aux critères de conception définis dans le service, mais qui pourraient être néanmoins jugées tout à fait acceptables dans un autre contexte de développement informatique ;
- Le nombre de *solutions incorrectes* ; c'est-à-dire celles qui fournissent des résultats inutilisables ;
- Le nombre de solutions *valides (ou satisfaisantes)*, c'est-à-dire celles qui donnent un résultat acceptable permettant de progresser efficacement dans l'activité.

Cette première approche quantitative des solutions sera le moyen d'apprécier la performance du processus de résolution d'après le type d'expertise (sur environnement site-central ou client serveur) et selon le niveau d'expertise considéré (novice ou expert). Un indice de "*rendement cognitif*" peut d'ailleurs se calculer selon le rapport :

$$\frac{\text{"nombre de solutions satisfaisantes"}}{\text{"nombre total de solutions exprimées"}} .$$

Ainsi, un informaticien aura un rendement cognitif optimal lorsque la totalité des solutions exprimées est exploitée en l'état. A l'inverse, un faible rendement cognitif signifiera qu'une majorité de solutions s'avèrent inutilisables car insatisfaisantes. Elles peuvent être alors inappropriées, incorrectes ou incomplètes.

2. En ce qui concernent les aspects **qualitatifs** des solutions, il s'agira de déterminer :
   i. la nature des problèmes rencontrés ;
   ii. la nature des stratégies de résolution mises en œuvre ;
   iii. les processus de sélection et d'évaluation des solutions ;
   iv. les types de traitements mis en œuvre (de "*haut en bas*" ou de "*bas en haut*", c'est-à-dire respectivement dirigés par les concepts ou dirigés par les données).

En définitive, l'ensemble de ces données seront analysées selon trois types de finalité :

i. étudier les stratégies de mise en œuvre, d'évaluation et de sélection des solutions ;
ii. comparer les solutions retenues par les différents sujets ;
iii. appréhender et expliciter les erreurs que les informaticiens novices commettent.

3.2.2 Aspects quantitatifs des solutions déployées durant l'activité de maquettage

3.2.2.1 Présentation des résultats

Le tableau ci-dessous présente le nombre moyen de solutions formulées par les informaticiens, pour chaque type de solutions considérées.

		5 Expert Pacbase	5 Expert NSDK	5 Novice NSDK
Nombre de solutions exprimées en moyenne par chaque développeur durant l'activité de maquettage		29	56	87
Parmi lesquelles :				
Solutions Insatisfaisantes	***Nombre de solutions incomplètes***	2	5	16
	Nombre de solutions inappropriées	1	12	10
	Nombre de solutions incorrectes	3	4	20
Solutions satisfaisantes	***Nombre de solutions valides***	23	35	41
	Indice de rendement cognitif *(nombre de solutions valides / nombre total de solutions exprimées) * 100*	80%	62%	47%

Figure 61
Résultats du modèle de résolution : aspects quantitatifs des solutions

Une première remarque concerne d'abord le nombre moyen de solutions exprimées par chaque échantillon de développeurs. Il n'y a aucune homogénéité entre ces chiffres. Le nombre de solutions est ainsi plus élevé chez les novices *NSDK (87 solutions exprimées)* que chez leurs collègues experts en NSDK (56) ou en Pacbase (29).

Une deuxième remarque concerne la pertinence des solutions proposées, ou plus exactement, leur adéquation à la tâche ou au problème rencontré. On se rend compte que ce sont les novices NSDK qui proposent les solutions les moins appropriées, avec un indice de rendement cognitif assez faible de l'ordre de 47%, alors que les experts Pacbase ont un excellent indice (près de 80%).

Une troisième remarque porte sur le nombre de solutions incomplètes : il manque des éléments qui permettraient de fournir d'emblée une solution acceptable. Ce type de solution est assez rare chez les experts, qu'ils soient NSDK (5) ou Pacbase (2), alors qu'il est plus fréquent chez les novices (16).

Enfin, la catégorie des novices est l'échantillon qui propose le plus grand nombre de solutions de maquettage incorrectes (20). A l'opposé, le groupe des experts Pacbase et NSDK en formule assez peu (respectivement 4 et 3).

Nous allons à présent revenir sur tous ces résultats en analysant d'abord, les données de ces trois échantillons, puis, en apportant des explications quant aux différences observées.

3.2.2.2 Analyse des résultats

a) Solutions exprimées par les experts Pacbase

Le nombre de solutions exprimées par cet échantillon est non seulement le plus faible, mais de surcroît, les solutions mises en œuvre se révèlent presque toutes correctes (rendement cognitif de 80%).

> *« Tu vois ici, je veux réaliser un écran fiche* [pour la saisie, NDR] (But). *Mais je sais que cet écran est assez typique de ce que l'on fait habituellement. Alors là, je consulte donc la doc* (Solution 1) *pour l'identifier et surtout retrouver son emplacement dans la bibliothèque des entités. Ca y est. Donc là, je tape la commande pour accéder aux écrans de recherche* (Solution 2) , *puis le code de l'écran en question et il s'affiche* (Solution 3). *Je vérifie quand même qu'il correspond bien à ce que je recherche, et si c'est bon je vais l'insérer* (Solution 4*) dans mon projet»*

Pour cette procédure de récupération d'écran, l'informaticien déploie par exemple une procédure de résolution comprenant quatre actions élémentaires.

Ce rendement élevé peut s'expliquer par le fait que la réutilisation fréquente des solutions stockées dans les sources internes –déjà testées et "homologuées" sur les anciens projets– d'une part, et les demandes récurrentes des utilisateurs d'autre part, donnent la possibilité à l'informaticien de spécifier d'emblée des solutions fiables et complètes. Des rejets peuvent cependant apparaître lorsque le développeur présente une solution qui ne correspond pas exactement aux canevas de conception de Pacbase (squelette ou grille de saisie). Dans ce cas, l'informaticien sera conduit à revoir localement ou globalement sa solution pour l'adapter aux prescriptions des trames de développement. Les cas les plus fréquents sont observés lors de la définition des coordonnées de positionnement des rubriques : une erreur de quelques centimètres et c'est tout l'écran qui se trouvera bouleverser.

On notera enfin que les solutions sont généralement définies en amont de la réalisation, c'est-à-dire pendant l'analyse organique, et qu'elles sont appliquées en l'état, sans révision, durant la conception.

b) Solutions exprimées par les experts NSDK

Le nombre de solutions est logiquement plus élevé sur NSDK que sur Pacbase puisque les tâches et les buts à atteindre sont plus nombreux. En effet, le modèle d'action a montré que l'informaticien planifiait un plus grand nombre de sous-buts à cause de la double tâche de conception-programmation de l'interface. Il lui faut donc exécuter plus d'actions pour concevoir :

i. les aspects graphiques de la maquette ;

> *« bon maintenant, je vais construire une fenêtre de validation, demandant à l'utilisateur de confirmer l'enregistrement des données qu'il vient de saisir et qui vont écraser les anciennes* (but). *Donc je choisis cette fenêtre de communication* (solution 1) *que j'amène sur le plan de travail [petite fenêtre de dialogue, NDR], voilà. Je lui donne un nom qui sera "Validation de l'enregistrement"(...)* (Solution 2). *Je mets aussi du texte dans cette zone de commentaire "Voulez-vous remplacer les données existantes ? " (...)* (Solution 3). *Je déplace les trois contrôles dont j'ai besoin de cette barre (boîte à outils) à la fenêtre* (Solution 4). *Je nomme par exemple celui-ci en cliquant sur le bouton droit de la souris, une boîte de dialogue s'affiche et je mets le nom dans cette zone : "Oui" (...)* (Solution 5)»
>
> Dans ce cas, cinq solutions sont enchaînées par l'informaticien pour composer graphiquement une fenêtre.

ii. les caractéristiques internes de la maquette (la programmation) ;

> *« En fait, en fonction du fonctionnel, on va choisir certains types de fenêtres pour certains affichages. Là, par exemple, je veux une fenêtre dialogue où l'utilisateur n'a pas d'autres choses à faire que de faire "Ok" ou "Annuler"* (but). *Il ne pourra pas aller s'amuser à cliquer sur le menu : il sera bloquer* [prérequis]. *Donc je vais sélectionner une fenêtre modale* (solution 1) *et lui associer un événement get-locus* (solution 2)»
>
> Ici, le novice détermine deux solutions pour programmer une fenêtre modale.

Une autre caractéristique de cet échantillon est qu'il présente un taux de solutions insatisfaisantes assez élevé, de l'ordre de 21 pour 35 solutions exprimées (indice de rendement cognitif de 62%). Les compétences des informaticiens (mauvaise

représentation du problème, capacité cognitive limitée, expériences insuffisantes ou peu fiables...) ne sont pourtant pas incriminées. Les causes sont plutôt à rechercher du côté de la diversité de leur expérience. En effet, ces experts, qui sont aussi des prestataires extérieurs, réutilisent des procédures de maquettage qu'ils avaient l'habitude d'appliquer lors de missions antérieures. Seulement, ces solutions ne conviennent plus aux critères de conception définis par la charte de développement de DABFI. Près de 12 solutions inappropriées sont ainsi exprimées en moyenne par cette catégorie d'informaticiens. En somme, si les solutions rejetées sont *contextuellement* inappropriées par rapport aux normes de développement du service, *conceptuellement*, elles restent tout à fait acceptables.

> *« Il y a des normes données par la charte graphique. Par exemple, les boutons, c'est soit en haut à droite ou en bas à droite, en fonction des champs qui buttent dedans. Mais des fois, je m'aperçois que la mise en forme d'une fenêtre* [positionnement des boutons sur la fenêtre, NDR] *correspond à un projet que j'avais réalisé dans une mission précédente. Le problème c'est que DABFI-O n'est pas d'accord pour accepter ce qu'on propose, même si ça fonctionne »*

Une autre explication est à mettre en perspective avec le modèle d'action que déploient ces informaticiens expérimentés. Leur plan de maquettage est en effet constamment modifié dans le cadre de la conception pour tendre vers une gestion plus située de l'activité. Les solutions qui, à un instant "*t*", étaient appropriées au plan de conception peuvent ne plus l'être, à un autre moment, parce que ce plan et ses objectifs ont évolué. L'informaticien doit donc faire progresser ses solutions en même temps qu'il élabore de nouvelles versions du plan d'action. C'est pourquoi, une solution peut subitement devenir inadaptée si le but pour lequel elle avait été initialement élaborée change en cours d'action.

Pour terminer, un nombre important de solutions est défini dynamiquement durant l'étape de réalisation de la maquette. Rappelons que sur Pacbase, elles étaient plutôt élaborées dès l'analyse organique. Ce mode de fonctionnement se justifie par les interactions continues qui se déroulent entre le mode d'organisation contextualisé de l'activité d'une part, et les résultats de la résolution d'autre part : l'expert est amené à redéfinir ses buts en fonction des progrès de la résolution, et inversement, à trouver de nouvelles solutions lorsque les objectifs changent.

c) Solutions exprimées par les novices NSDK

Si les solutions proposées par ces novices sont de loin les plus élevées parmi les trois échantillons considérés, il n'en reste pas moins que leur rendement cognitif est le plus faible (47%) à cause des nombreuses solutions insatisfaisantes formulées (qu'elles soient incomplètes, incorrectes ou inappropriées). Les deux paragraphes qui suivent vont fournir des explications à ce phénomène.

1. Les solutions incomplètes se caractérisent par l'absence d'un certain nombre d'éléments dans la solution permettant de résoudre le problème et d'atteindre le but souhaité. Les solutions incorrectes produisent des solutions inutilisables. Ce mode inachevé de résolution résulterait en fait d'une définition approximative des **critères** et des **contraintes** de conception par l'informaticien novice.

   Parmi ces **critères**, on peut distinguer (à partir de la description de Bonnardel, 1991) :

   - les *critères de fonctionnalité* qui décrivent la fonction attendue du produit à concevoir et qui sont désignés dans l'énoncé ou les spécifications du problème ;
   - des *critères d'évaluation* qui sont utiliser pour porter un jugement sur les solutions.

   Dès lors, l'informaticien propose des solutions insatisfaisantes lorsque *i)* il n'arrive pas à identifier précisément les états souhaités de la maquette (*critères de fonctionnalité*) et/ou lorsque *ii)* les critères d'évaluation utilisés sont mauvais. Deux exemples illustrent ces cas.

   i) Le critère "*d'accessibilité à l'interface*" a par exemple été formulé par l'utilisateur dans le cahier des charges. Pourtant, ce *critère de fonctionnalité* n'est pas jugé prioritaire par l'informaticien. Cette négligence le conduit à réaliser une fenêtre sans titre, dépourvue de commandes rapides et d'aide en ligne, et avec très peu de boutons de navigation. Cette solution a finalement été rejetée parce que la maquette n'était pas assez conviviale.

ii) Dans un second cas, les "*critères d'évaluation*" manquent de finesse pour évaluer la pertinence d'une solution. Ils laissent ainsi passer des solutions qui se révéleront finalement incorrectes ou incomplètes dans le cadre de l'activité. Parmi ces "*critères d'évaluation*", on peut distinguer :

* les "évaluateurs graphiques" :

  Dans cet exemple, l'informaticien expose différentes exigences de conception à respecter :

  « Quand je vais construire la maquette, il faudra que je fasse attention à différentes choses : limiter le nombre de fenêtres qui peuvent s'afficher simultanément sur un écran pour ne pas alourdir la présentation ; respecter le nombre d'options à proposer dans un menu et le nombre d'items à présenter dans ces options ; veiller à ce que la dénomination des boutons et des fenêtres soit proche du vocabulaire métier de l'utilisateur »

* des "évaluateurs logiques" :

  Ici, l'informaticien cherche à contrôler la cohérence entre différents événements d'un même objet (Cf. extrait de corpus plus bas), à évaluer les effets contextuels d'un événement, à estimer le temps de réponse de l'interface…

  « (...) Je vais prendre celui-là : les liens de cette fenêtre avec le système aval. Par exemple, lier, c'est avec un bouton. Je ne comprends pas... ah oui. Le bouton "OK" est un contrôle. Au niveau des événements, c'est de la programmation événementielle, donc, j'ai plusieurs événements : et si j'exécute la fonction OK, il faut faire attention que son événement ne donne pas un comportement qui pourrait me bloquer. Il faut donc simuler ce bouton. »

Par ailleurs, les **contraintes** de conception sont susceptibles de générer des solutions insatisfaisantes. Ces contraintes représentent des conditions à satisfaire pour obtenir le but, et constituent de ce fait des guides pour la résolution. Les informaticiens les déterminent à partir du cahier des charges, des critères de fonctionnalité (vus plus haut) et de données externes au projet (planning, budget, ressources humaines disponibles). Les solutions non conformes proviendraient de l'établissement de contraintes trop "*rigides*" ou, au contraire trop "*floues*" :

i) dans le cas de "*contraintes rigides*", l'informaticien s'enferme dans un cadre de résolution trop étriqué qui l'empêche d'imaginer des solutions plus appropriées ;

> *« DABFI-O nous bassine toujours avec ce qu'ils appellent la lisibilité de l'interface. C'est pour cela que je me limite à 2 ou à 3 boutons sur une fenêtre et que je leur donne, dans la mesure du possible, pas plus de 3 événements. Comme ça, ça limite les risques d'encombrement" »*

ii) dans le cas de "*contraintes floues*", c'est l'inverse qui se produit. Les repères dont dispose l'informaticien ne sont pas assez précis pour fixer des objectifs et organiser la résolution. La solution proposée est alors trop générale et ne peut être appliquée en l'état. Il faut alors rajouter de nouvelles contraintes pour l'affiner et l'exécuter.

"Pour des questions de coût et de fiabilité de l'interface, je vais essayer de limiter le nombre de fenêtre".

Mais quel coût ? Quel type de fiabilité recherchée ? Combien de fenêtres ? L'informaticien ne le précise pas.

2. Les solutions désignées comme ***inappropriées*** ont une toute autre origine. Elles résultent de l'utilisation inadéquate de schémas de résolution récupérés du contexte de conception site-central par un transfert analogique négatif. Nous allons en présenter deux illustrations à travers respectivement de : (*i*) *la composition graphique* et (*ii*) *la programmation de l'interface.*

   i. *La composition graphique de l'interface*

   Au cours de nos observations, nous nous sommes rendus compte qu'une erreur fréquente chez les informaticiens débutants consistait à centraliser les commandes de l'application :

   * *soit sur la zone fenêtre* ; on propose un maximum de boutons sur la fenêtre (un bouton par fonction de commande) (*Cf. Figure 62*)
   * *soit sur la zone menu* ; toutes les commandes se retrouvent rassemblées au niveau du menu de la fenêtre (*Cf. Figure 63*).

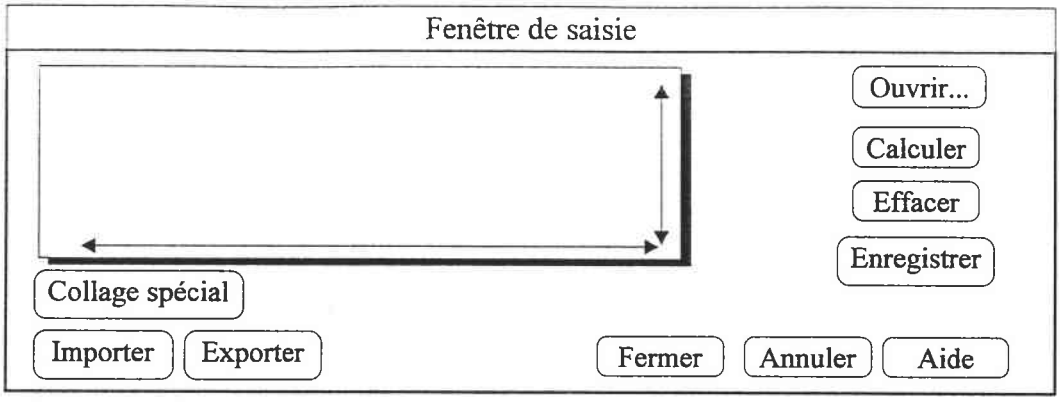


Figure 62
Exemple où la plupart des commandes sont centralisées sur la zone fenêtre, sous forme de boutons

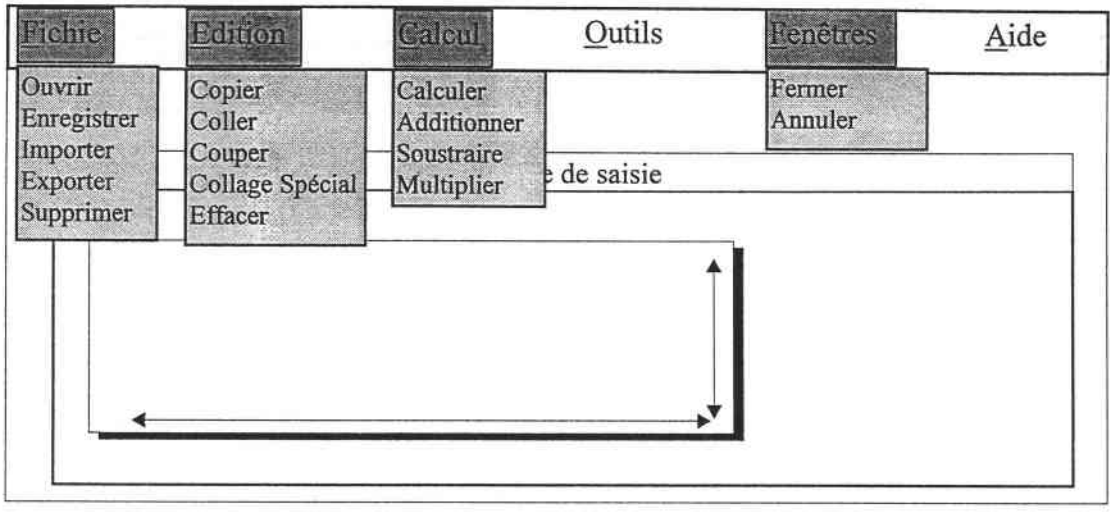


Figure 63
Exemple où la plupart des commandes sont centralisées dans la zone "Menu"

En fait, ces deux stratégies de maquettage proviennent de la conception site-central. En effet, les informaticiens opérant sur Pacbase ont pour habitude de multiplier le nombre de commandes disponibles sur un même écran afin de réduire la taille de l'application. Cette pratique de rationalisation de l'affichage est donc conservée par les débutants qui regroupent les commandes sur des zones déterminées de la fenêtre (soit sur le menu, soit sur la fenêtre).

> *« Donc là* [sur NSDK, NDR], *on s'est complètement trompé puisqu'on a fait des fenêtres qui sont énormes avec plein d'informations. Et je ne m'en suis pas rendu compte immédiatement, mais cela correspond apparemment vraiment à une logique site-central où on fait tout en même temps sur la fenêtre ».*

ii. Programmation de l'interface

Une autre difficulté concerne la programmation des événements de la maquette, et plus particulièrement la gestion de leurs effets. On rappelle que deux types d'effet peuvent être générés par les événements sur le reste de la maquette :

- soit un ***effet local***, c'est-à-dire que l'action de l'événement reste limitée à la fenêtre où il est déclenché ;

  «Dans ce cas, si l'utilisateur sélectionne le bouton "Annuler", j'ai programmé sa fonction de telle façon qu'il annule les modifications effectuées sur cette fenêtre seulement ».

- soit un ***effet global et contextuel*** (effet de bord), c'est-à-dire que son action peut se propager au reste de l'interface et générer des modifications sur des composants non affichés de la maquette.

  « (...)"Alors qu'ici, la sélection du bouton "Calculer" implique plusieurs actions : démarrer le calcul, ouvrir une seconde fenêtre où l'on présente des courbes, fermer la fenêtre des tableaux, et griser certaines options du menu qui ne sont plus disponibles pour l'utilisateur. Donc là, les événements et leur programmation sont plus complexes qu'avant. Il faut penser à tout. »

Sur la base des observations réalisées, nous avons constaté que la prévision de ces "effets de bord" posait de gros problèmes aux informaticiens néophytes. Les solutions préconisées étaient toutes bâties sur le modèle de la gestion locale des événements du site-central :

« Quand on travaille sur une fenêtre ici [Sur NSDK], *on n'est pas lié qu'à cette fenêtre, on a tout le contexte autour qui peut bouger, qui peut évoluer... Alors qu'en site-central, lorsque l'on travaillait sur une fenêtre, c'était plein écran... ce qu'il y avait avant ne comptait pas. C'est pas facile d'évoluer, je m'embrouille les pinceaux assez fréquemment».*

Dans ce cas de gestion des événements, comme dans celui de la composition graphique, c'est la réutilisation d'un mode de raisonnement anachronique qui induit les erreurs de conception.

En définitive, il ressort que les stratégies de résolution du débutant sont encore insuffisamment matures pour être employées convenablement durant cette étape de maquettage. C'est parce qu'il est cognitivement démuni devant de nouveaux problèmes (graphiques ou logiques) que l'informaticien a tendance à faire spontanément appel aux solutions qu'il connaît et maîtrise le mieux, à savoir celles issues du site-central. Il existerait ainsi une sorte de recouvrement des anciens schémas de résolution sur les nouveaux, causé par des transferts d'apprentissage inappropriés.

Nous aurons l'occasion de revenir plus en détails sur ces transferts négatifs dans l'approche qualitative des solutions qui va dès à présent être exposée.

3.2.3 Aspects qualitatifs des solutions déployées durant l'activité de maquettage

3.2.3.1 Présentation des résultats

Ce tableau présente un descriptif des différentes stratégies de résolution qui ont été identifiées chez les informaticiens.

	5 Experts Pacbase	5 Experts NSDK	5 Novice NSDK
– ***Nature des problèmes à gérer***	Proche d'un problème de transformation d'état	Problème de conception	
– ***Représentation du problème***	Stable	Versatile	
– ***Nature des stratégies de résolution mises en œuvre***	Récupération analogique de solutions	Construction de solutions	Mixte des deux : Récupération anachronique de solutions et construction de solutions
– ***Processus de sélection et d'évaluation des solutions***	En largeur d'abord	En largeur d'abord, mais aussi En profondeur d'abord.	En profondeur d'abord
– ***Moment de régulation et d'ajustement des solutions***	En début d'activité	Au cours de l'activité	Au cours de l'activité
– ***Type de traitements mis en œuvre***	Dirigés par les concepts (Top-Down)	Dirigés par les données (Bottom-up)	Mixte des deux : Dirigés par les données et les concepts

Figure 64
Résultats du modèle de résolution : aspects qualitatifs des solutions

Si l'on observe l'ensemble de ces résultats, on se rend compte que les conduites cognitives des informaticiens experts Pacbase et NSDK différent de manière assez significative, notamment sur les façons de se représenter un problème, de trouver une solution, de la sélectionner, de l'ajuster, et de déclencher un traitement.

On remarque également que les informaticiens novices sont plus souvent partagés entre deux fonctionnements cognitifs, en particulier lorsqu'ils alternent ou combinent des procédures de résolution, soit récupérées de la conception site-central, soit développées spécialement pour la conception client serveur.

Enfin, la nature même des problèmes à résoudre est différente selon qu'ils sont posés dans un environnement site-central ou client serveur. Dans le cadre de la programmation site-central, c'est un problème routinier dont la structure s'apparente à celle des problèmes de transformation d'état. Dans l'autre contexte (client serveur),

nous sommes en présence d'un problème non familier de conception. C'est ce qui expliquerait d'ailleurs, que la représentation du problème soit dans un cas stable (site-central) et dans l'autre cas versatile (client serveur).

Reprenons et détaillons chacun de ces résultats en commençant par le type de problème rencontré.

3.2.3.2 Analyse des résultats

3.2.3.2.1 Nature des problèmes rencontrés

Dans notre cadre théorique, nous avions relevé que les chercheurs considéraient la tâche informatique comme une activité de résolution de problème de conception à part entière, parce que l'état initial est mal défini et l'état final est à définir. L'activité de conception consiste alors à déterminer le problème et sa solution.

Toutefois, sur la base des analyses effectuées, il ressort que le contexte particulier de la conception site-central induit une structuration des problèmes qui relèverait davantage de celle des problèmes de "transformation d'état". En voici les raisons.

a) Les problèmes rencontrés en développement site-central appartiennent à la catégorie des problèmes de transformation d'état

Dans le cadre du développement site-central, les très faibles marges de manoeuvre laissées à l'informaticien, aussi bien dans le design de l'interface que dans la programmation logique de l'application, limitent la fabrication d'applications réellement innovantes. Les informaticiens sont ainsi enfermés dans une sorte de modèle standard de conception informatique qu'ils s'évertuent à reproduire à chaque projet. Ils sont d'autant plus poussés à le faire que les demandes des utilisateurs se renouvellent peu à cause des limites techniques du site-central. Outre ce cadre de développement passablement sclérosé, les informaticiens ont la possibilité de récupérer des solutions déjà toutes prêtes d'anciens projets. Cette conception s'apparente dès lors plus à du "*prêt à porter*" informatique qu'à du "*sur mesure*".

> *« On se rend compte tout de suite dans l'analyse organique* [étape de description et de représentation du problème] *qu'il y aura telle requête qui sera difficile, vu que l'on est confronté souvent aux mêmes demandes ».*

Pour ces raisons, les problèmes posés en site-central se rapprocheraient donc des problèmes de transformation d'état. En effet, d'après la définition qui en est donnée,

(Falzon, 1995), l'individu doit trouver un "chemin", une méthode permettant de ramener une situation initiale insatisfaisante (ce qu'il connaît) à une situation cible désirée (qu'il connaît aussi). Pour cela, il dispose d'un certain nombre de moyens d'actions (opérateurs de transformation) et doit respecter les contraintes sur les états intermédiaires (certains états sont interdits ou peu souhaitables).

Si l'on applique cette description aux problèmes rencontrés dans le site-central, on retrouve là les mêmes caractéristiques : les états initiaux et cibles du problème sont identifiés dès le début du développement (grâce à la récurrence et à la familiarité des projets). Quant aux moyens d'action, ceux-ci sont obtenus par l'intermédiaire du dictionnaire de Pacbase. Enfin, les contraintes sur les états intermédiaires sont parfaitement identifiées puisqu'elles découlent des prescriptions issues des canevas de conception de Pacbase.

> *« Notre activité* [de conception avec *NSDK*, NDR] *c'est vraiment de l'exploration , c'est pratiquement de l'expérimentation , tu vois. C'est typique. Contrairement à une application classique site-central où l'on sait tout de suite ce que l'on veut et ce que l'on va faire».*

b) Les problèmes rencontrés en développement client serveur sont typiquement des problèmes de conception

En effet, la puissance de l'outil permet de répondre à pratiquement toutes les demandes de conception. Les utilisateurs le savent et exigent des développements différents et toujours plus innovants. De plus, le stock de projets n'étant pas aussi abondant que dans Pacbase, l'informaticien est obligé de créer de toute pièce sa solution pour chaque nouveau projet. Enfin, il peut exister plusieurs solutions admissibles pour un même problème : l'outil est en effet assez souple pour accepter différents modèles de résolution. Dans ces conditions, l'informaticien se trouve effectivement dans le cas de la résolution de problème de conception.

3.2.3.2.2 La représentation du problème

Compte tenu du type de problème à résoudre, les représentations de l'informaticien sont :

a) stables sur le site-central ;

b) versatiles en client serveur.

a) Une représentation stable du problème sur site-central

La conception est une opération qui reste relativement stable et bien maîtrisée sur l'environnement site-central. Rappelons brièvement quelques caractéristiques de cette situation qui ont déjà été identifiées : *i)* les problèmes posés varient peu, *ii)* les déviations opportunistes du plan de développement sont extrêmement rares, *iii)* les trames de conception sont figées, et *iv)* les difficultés rencontrées sont assez vite identifiées et résolues. En conséquence, les représentations que l'informaticien se construit du problème, de sa solution et du plan de conception ont donc très peu de chances d'évoluer.

b) Une représentation labile et versatile sur le client serveur

Dans le contexte de développement client serveur, l'informaticien doit au contraire modifier régulièrement sa représentation du problème parce que la situation de conception évolue constamment. C'est pourquoi, à cause des nombreuses opportunités de conception qui apparaissent durant le maquettage, en raison aussi des contraintes qu'il faut satisfaire, ou bien encore de la redéfinition régulière des objectifs, la conception en client serveur est par essence une situation instable.

> *« Les goulots d'étranglements, tu en as identifiés certains sur le papier. Et puis, alors là, tu vas en avoir un paquet qui vont se révéler à l'implémentation, à l'exécution auxquels tu n'as pas pensé. La plupart du temps, ils vont être mineurs, puis tu vas tomber sur le majeur (...) Il faut alors tout repenser le problème».*

La représentation du problème se modifie donc au gré des évolutions de la situation de conception. Il suffit que ce contexte change, ou qu'un élément non remarqué de la situation soit pris en compte, alors qu'il ne l'était pas, pour que la représentation se modifie.

> *« Quand on réalise, on se rend compte que la vision du problème qu'on a avait au départ est remis en cause: elle évolue. Même, l'application qu'on réalise n'a habituellement rien à voir avec l'application qu'on avait imaginée »*

c) En résumé

Deux types de représentation se détachent donc selon le contexte de conception :

i. Sur l'environnement site-central, la représentation du problème est **stable** parce qu'elle se base sur la structure du squelette et la régularité du contexte de conception (problèmes récurrents, contraintes identiques, peu d'opportunités de conception offertes et déviation du plan quasi impossible).
ii. En environnement client serveur, la représentation du modèle de conception sera plutôt **instable et versatile** car elle s'actualise au fur et à mesure des découvertes et des circonstances de la résolution.

3.2.3.2.3 Processus de résolution

Les processus de résolution identifiés dans la conception informatique divergent selon qu'ils se déroulent dans un environnement site-central ou client serveur et, selon aussi qu'ils sont élaborés par une population d'experts ou de novices. La résolution sera alors :

a) analogique pour des experts sur un environnement Pacbase
b) exploratoire dans le cas d'experts NSDK
c) et une combinaison de ces deux stratégies dans le cas des novices NSDK.

a) Une construction analogique des solution chez les experts Pacbase

Dans le cadre de conception procédurale de Pacbase, la démarche de résolution est analogique. Elle s'associe habilement aux méthodes structurées et aux canevas de Pacbase. Le développeur part d'une fonction générale identifiée dans la représentation du problème et la raffine successivement durant l'analyse organique, pour déterminer les concepts à implémenter : données, règles de gestion, fonctions, etc. Le concepteur crée ainsi un modèle abstrait décrivant toute l'application en termes d'une ou plusieurs notations (organigrammes, flux de données, pseudo-codes...).

L'informaticien s'appuie sur cette décomposition du problème pour, d'une part, évoquer des sous-fonctions familières et, d'autre part, retrouver les plans de résolution appropriés stockés dans les sources de connaissances externes (dictionnaire d'entités) ou internes (mémoire) :

« Convertir un cours en francs peut-être un algorithme relativement complexe. Si l'on cherche à décomposer cette fonction on peut avoir deux possibilités :

d'abord travailler sur le mot clef convertir, et faire une recherche de sous-programmes dans la bibliothèque. Après, si cela ne donne rien, c'est de se dire que la conversion peut se faire sur la rubrique cours. On fait alors la même chose ».

Durant la résolution, l'informaticien poursuit en fait toujours le même dessein : s'assurer que tout projet, même inhabituel, renferme des sous-parties équivalentes à, ou jugées très proches de celles habituellement traitées. En cela, la décomposition par raffinements successifs fournit le moyen d'identifier des régularités conceptuelles entre de nouveaux problèmes et l'espace problème traditionnellement évoqué par l'utilisateur :

« il y en a beaucoup qui sont pareils [de problèmes, NDR] *(...), ce sera toujours le même principe un peu partout et après, il y a beaucoup de choses qui se retrouvent ».*

D'ailleurs, le caractère récurrent des projets permet de caractériser assez rapidement les problèmes familiers et de repérer des solutions pertinentes présentes dans le dictionnaire de Pacbase. Leur mise en oeuvre dans le plan de conception se déroule généralement sans modification profonde. Les principaux changements portent sur les parties de la solution qui ne correspondent pas au domaine à traiter. L'informaticien décompose alors la solution et intervient sur les niveaux à modifier :

« On recherche d'abord un programme qui ressemble à ce qu'on veut faire, et à partir de ce programme là, vous pouvez tout à fait le réutiliser tel quel, si jamais il correspond à ce que vous recherchez. Autrement vous pouvez le copier et faire quelques modifications» .

En somme, le processus de résolution consiste à dériver des solutions récupérées de projets antérieurs par un double mécanisme :

1. l'informaticien définit d'abord les solutions d'un premier niveau qui se situent à un niveau physique grâce à la décomposition fonctionnelle. Elles concernent des choix techniques de réutilisation tels la sélection des écrans, des fonctions, etc. récupérables dans le dictionnaire ;
2. puis, il détermine des solutions de second niveau qui se situent à un niveau conceptuel. Celles-ci concernent l'élaboration de solutions schématiques englobant les différentes solutions récupérées.

Finalement, la démarche de résolution de l'informaticien se base sur la **réutilisation** de solutions préexistantes, et par la recherche **d'analogies** avec des situations de conceptions passées

b) Une construction dynamique des solutions chez les experts NSDK

Dans ce cadre de conception événementielle, les stratégies de résolution utilisées divergent de celles employées par les experts Pacbase. Le concepteur tente d'élaborer une première solution conceptuelle (compte tenue de l'idée qu'il a de l'interface à concevoir) qu'il va progressivement préciser (de manière itérative et exploratoire) en adoptant des points de vue de plus en plus concrets (changement de représentation de la solution) ; jusqu'à finalement parvenir à la définition des caractéristiques physiques et techniques de la solution.

Pour réaliser efficacement ces différentes étapes de raisonnement et évaluer la pertinence de ses solutions, l'informaticien se livre à une construction dynamique et incrémentale de la solution, *via* différentes stratégies :

i. la construction descendante, puis ascendante
ii. la construction évolutive,
iii. la construction comparative,
iv. la construction par validation.

i. La construction descendante, puis ascendante

L'informaticien dispose d'une première représentation de solution qui reste toutefois trop générale et trop complexe pour être traitée en l'état. Il va alors la raffiner progressivement en fonctions élémentaire (démarche descendante), puis reconstruire, par un processus de mise en œuvre, des "blocs" logiques de solution, en procédant à des ajouts et/ou à des modifications de fonctions, de données ou d'objets graphiques (démarche ascendante). Il établit également des relations logiques entre ces blocs : par exemple, une fenêtre principale associée à la fois à une boîte de dialogue de saisie et à une fenêtre de commentaire.

« Disons, que ce qui est intéressant là dedans, c'est qu'on va découper en fonctionnalités élémentaires. Donc par exemple, créer un processus fédéral, le modifier, etc. Il va falloir avant que l'on crée toutes ces fonctionnalités élémentaires, toutes celles qui ont un lien avec l'interface (...) Après on pourra définir des boîtes logiques, définir des liens entre ces entités pour créer des enchaînements logiques»

ii. La construction évolutive

Plus le développeur avance dans la conception, plus la représentation du problème se modifie, s'affine et se précise. La version de la solution va donc s'ajuster graduellement à l'évolution de la représentation du problème. Tout se passe en fait comme s'il y avait un processus d'adaptation dynamique des solutions au contexte de travail, par lequel l'individu teste, corrige et améliore ses stratégies de résolution. Cette démarche cognitive se rapproche d'une stratégie de type fin-moyen où l'individu a la possibilité d'évaluer à tout moment l'écart entre l'état actuel (solution) et l'état final souhaité (buts):

« J'ai une idée assez précise, avant de commencer à programmer, des classes, des fonctions et de l'architecture du système. C'est l'implémentation de ce modèle et effectivement son test qui va valider ou infirmer cette idée et la modifier »

iii. La construction comparative

La sélection d'une solution pertinente parmi plusieurs possibles est également une caractéristique importante de la conception client serveur. Il se peut en effet que l'individu dispose de plusieurs stratégies admissibles pour un même problème. La construction comparative consiste alors à mesurer l'impact fonctionnel et technique de chacune des solutions (par une stratégie "*en largeur d'abord*" ; nous aurons l'occasion d'y revenir ultérieurement). La difficulté n'est plus alors de trouver une solution mais de définir celle qui semble la plus appropriée au contexte de conception :

« Ce qu'il y a en général, quand c'est un truc nouveau que l'on expérimente, je vois que j'ai pu trouver 2 à 3 façons de résoudre le problème. Alors là, j'essaie d'évaluer quelle est la meilleure : soit en facilité, en général je le sais déjà, en efficacité, en qualité de résolution. Après, quand tu as un choix à faire, il faut en retenir une ».

iv. La construction par validation (heuristique)

Cette dernière démarche doit permettre à l'informaticien de vérifier qu'une intuition de conception peut lui fournir une trame intéressante de résolution. Cette stratégie s'apparente à celle que Varela (1990) a défini dans sa théorie de l'énaction et qu'il nomme "*stratégie émergente*". Elle représente des actions *a priori* hétérogènes qui, entreprises une par une, vont progressivement converger vers une sorte de cohérence, menant vers des voies de solutions prometteuses.

En ce sens, la stratégie par validation aurait une fonction heuristique qui favoriserait la découverte de solutions par association et enrichissement progressif des idées :

« Tu as une idée, tu dis "tiens", il faudrait faire comme cela: tu l'essaies et tu vois si effectivement cela colle avec ce que tu avais prévu. Puis tu en essais une autre et une autre. Et à la fin, tu te rends compte que petit à petit, tu trouve quelque chose d'intéressant »

En définitive, il n'existe pas de solutions aussi clairement définies qu'en conception procédurale. Il y a au contraire une évolution active et opportuniste des schémas de résolution qui passe par des phases de tests, d'affinages successifs et d'application des solutions choisies.

c) Une stratégie hybride entre construction et récupération chez les novices NSDK

En passant de la conception site-central au développement client serveur, l'informaticien est donc passé d'une *situation de conception répétitive* – où il s'agissait de sélectionner les solutions idoines parmi un large panel disponible– à *une situation de conception novatrice* – où il s'agit dorénavant de construire des solutions originales–. De même, les logiques de conception divergent selon les contextes de développement : logique transactionnelle et séquentielle assujettie aux squelettes de programmation dans un cas (Pacbase), logique événementielle et asynchrone, affranchie du carcan technique dans l'autre cas (NSDK).

Confronté à de telles évolutions, le novice doit remanier ses structures cognitives pour proposer des solutions plus appropriées. Différentes stratégies ont été repérées.

1. Ainsi, lorsque le débutant considère que le problème rencontré est <u>*strictement identique*</u> à celui qu'il traitait sur le site-central, il réutilise spontanément les solutions provenant de son expérience passée. La rapatriement de ces schémas de résolution s'effectue alors par <u>raisonnement analogique</u> : le sujet transfère une procédure connue vers une situation nouvelle.

   Par exemple, l'informaticien fait souvent le rapprochement entre la démarche de conception graphique par manipulation directe (NSDK) et la démarche de maquettage dynamique sur Pacbase (par positionnement de rubriques sur un écran vierge) :

« Je dois constituer ma fenêtre en plaçant des contrôles, des boutons, des libellés, des ascenseurs avec la souris. Au début, c'est difficile à comprendre et à respecter. Mais je me dit que c'est la même chose que le "-L" de Pacbase [Maquettage dynamique] *lorsque je plaçais des codes sur un écran vierge et que j'appelais mes rubriques. Donc j'agis un peu de la même manière. »*

2. Lorsque l'informaticien considère le problème comme totalement *nouveau*, c'est-à-dire sans équivalent dans la conception site-central, il construit généralement les solutions en usant des mêmes stratégies que leurs collègues chevronnés ; à savoir des résolutions de type exploratoire, itérative, comparative et par validation. Nous en verrons des exemples plus concrets lorsque nous aborderons la procédure de désignation des objets graphiques de l'interface plus loin.

   Outre cela, il peut produire un raisonnement heuristique, basé sur des connaissances dont il dispose par ailleurs, pour élaborer une solution admissible.

   «Mon problème est de savoir ce que je mets sur le Changed, sur le selected, sur l'executed, [ce sont les événements de la maquette, NDR] *Je sais que j'ai du code à écrire, je connais le code, mais je ne sais pas derrière quel événement le mettre. Je sais que je dois programmer un contrôle de cohérence. Par contre où le placer, cela vient de la connaissance des événements possibles. Alors, évidemment, j'ai tendance à le mettre derrière Executed, parce que ça, au moins, cela s'exécute. J'en suis sûr. »*

   Dans l'exemple présenté, le problème qui se pose à l'informaticien est de déterminer l'événement qui supportera la fonction de "*contrôle de cohérence*" des données saisies. La solution envisagée est d'associer cette fonction à l'événement "*Executed*" car il serait, d'après les dires de l'informaticiens, systématiquement déclenché lors de la manipulation de l'interface.

   En fait, l'informaticien propose cette solution qui, conjointement à d'autres connaissances qu'il possède (sur les événements possibles et leurs propriétés), est compatible avec certains faits ou observations (ici, le déclenchement automatique de l'événement "*Executed*"). Cela dit, il s'agit d'une solution construite sur la base d'informations très limitées que de nombreux tests viendront confirmer et affiner, si nécessaire.

   « C'est de toute façon en testant qu'on voit si on n'a rien oublié, et on s'en rend très vite compte. C'est vite testé. C'est assez long à programmer, mais on voit au fur et mesure ce que cela fait, si cela se présente bien. »

3. Enfin, lorsque l'informaticien reconnaît certaines *similarités* entre le problème à gérer et une situation passée (au niveau de leur structure, de leurs données...), celui-ci est enclin à s'inspirer ou à partir de cette expérience passée pour élaborer des procédures de résolution. Cette analogie de "surface" est à l'origine d'un transfert par instanciation de schémas. On parlera alors plutôt d'évocation des schémas. L'informaticien se trouve dans le cas d'une démarche de résolution par transcendance d'apprentissage qui pourrait s'apparenter à de l'analogie, sauf qu'ici, l'individu n'applique pas, *ad litteram*, la procédure récupérée. Les schémas rapatriés subissent des adaptations.

   Par exemple, une difficulté courante en conception graphique concerne l'affichage *asynchrone* des fenêtres, c'est-à-dire que l'informaticien doit programmer l'interface pour que les fenêtres s'ouvrent en fonction des actions et des sollicitations de l'utilisateur. Cette cinématique événementielle s'oppose au mode *synchrone* (ou transactionnel) du site-central, qui détermine un enchaînement séquentiel et arbitraire des écrans. Les observations nous ont pourtant permis de constater que les novices construisaient leur maquette en n'utilisant pratiquement que des fenêtres de type modal. Or, ces fenêtres imposent un dialogue linéaire qui restreint les marges de manoeuvre de l'utilisateur final : il ne peut aller nul part tant qu'il n'a validé au moins une des commandes proposées par la fenêtre ("*Annulation*", "*Ok*"). Il doit alors suivre la logique prescrite par le système, et indirectement par l'informaticien.

   « Là, on s'aperçoit que quand on crée une fenêtre de ce type [modal, NDR], *qui est une fenêtre dialogue : l'utilisateur n'a pas d'autres choses à faire que de faire "Ok" ou "Annuler". Il ne pourra pas aller s'amuser à cliquer sur le menu : il va être bloqué »*

   En fait, l'informaticien reconstruit tout un dialogue transactionnel à l'aide des différents objets graphiques de NSDK.

   «Avec du recul, c'est vrai que je reprends cette logique du site-central. Je crois que je raisonne un peu trop comme sur Pacbase, c'est-à-dire un petit peu trop rigide dans l'affichage des fenêtres. (...) En fait, je ne laisse pas à l'utilisateur la possibilité de faire plein de choses».

   Dans ce cas, le développeur s'inspire de ses anciens schémas de maquettage pour développer une procédure de résolution : se basant sur la logique séquentielle d'animation des écrans de Pacbase, il élabore le dialogue de l'interface

graphique selon un mode conversationnel. Conceptuellement, cela revient à instancier les variables du schéma récupéré aux données de la nouvelle situation pour obtenir un plan de résolution opérationnel. On se trouve alors dans le cas d'un transfert d'apprentissage définit dans la théorie des schémas.

« Alors ça, c'est un peu comme sur le site-central, c'est-à-dire qu'il faut déterminer les critères obligatoires, les critères non obligatoires, est-ce que l'on veut deux ou trois lettres, faire des recherches génériques ou pas; ça c'est exactement pareil sur le site-central. Je dirais qu'au niveau de la conception à la limite c'est pareil. »

Cela dit, ce transfert est négatif puisque la procédure à laquelle l'informaticien se réfère n'est pas valide dans le nouveau contexte de conception. Aussi, bien que certains emprunts de l'expérience passée s'avèrent opportuns (*cas de la composition graphique des fenêtres par référence au maquettage dynamique*), il arrive également que la réutilisation conduise à des situations erronées (*exemple de l'enchaînement arbitraire des fenêtres*).

d) En résumé

La nature des stratégies de résolution mises en œuvre par ces novices dépendra des problèmes rencontrés. Elle sera :

i. tantôt **exploratoire** lorsque le problème est nouveau et que le répertoire de réponses de l'informaticien (extrait du site-central) ne regorge d'aucune solution jugée suffisamment pertinente pour résoudre celui-ci ;

ii. tantôt **analogique** s'il existe une solution qui peut être récupérée directement de son expérience site-central et être appliquée en l'état au problème client serveur (cas du raisonnement analogique) ;

iii. tantôt **heuristique** lorsque l'informaticien s'inspire de schémas de connaissances pour dépister de nouvelles voies de résolution. Ces transferts sont déclenchés à partir du degré de ressemblance que le sujet estime entre la situation problème et la situation passée déjà résolue.

3.2.3.2.4 Traitements mis en œuvre

Les données recueillies ont permis d'établir que :

a) la conception procédurale avec Pacbase était propice à la résolution dirigée par les concepts ;

b) les experts NSDK procédaient de manière opposée, c'est-à-dire selon une démarche déclenchée par les données de la situation problème,

c) les novices étaient partagés entre ces deux modes de résolution.

a) Traitements dirigés par les concepts chez les experts Pacbase

Ce traitement fait référence au fait que la connaissance que l'informaticien a déjà d'une situation problème, et les attentes qu'il a également, compte tenu de cette expérience, l'amènent à traiter plus rapidement certains problèmes que d'autres dans une situation donnée.

Pour être plus précis, ces traitements partiraient des représentations (stables) que l'informaticien a du problème pour aller vers la mise en œuvre des solutions provenant de ses sources internes (expériences) ou externes (dictionnaire d'entités).

On a par exemple observé que l'informaticien identifiait d'autant plus rapidement le type de problème à résoudre et la solution à mettre en œuvre que les demandes de développement émanaient d'utilisateurs familiers. Le développeur disposerait ainsi d'une sorte de cartographie mentale des exigences spécifiques à chaque groupe d'utilisateurs qui lui permettrait d'envisager d'emblée une solution. Il sait par expérience que tel service d'utilisateurs exigera tel type d'organisation d'écran, alors que tel autre groupe réclamera une autre composition.

> *« Lorsque les gens de la salle des marchés s'adressent à nous, on sait déjà ce qu'ils veulent et comment ils le veulent, à force de traiter avec eux. On devance même leurs besoins dans certains cas, vu qu'ils n'ont pas beaucoup de temps à nous consacrer »*

b) Des traitements dirigés par les données chez les experts NSDK

L'expression "***dirigés par les données***" ("*data-driven processing*") fait référence au fait que la perception d'un problème est d'abord fonction de ses attributs physiques élémentaires. Ce sont des attributs qui sont perçus lors de l'identification du problème. A chaque donnée, et donc à chaque informaticien identifiée, l'informaticien lui associe une connaissance mémorisée qui déclenchera un traitement particulier.

La conception en environnement graphique procure un contexte idéal pour ce type de raisonnement car son formalisme de représentation est très proche de la conception orientée objets. Il est ainsi possible de modéliser et de représenter directement les entités du monde réel en objets "graphique" informatiques. L'exemple le plus souvent cité est d'ailleurs la reproduction des documents de travail des utilisateurs finaux sur l'interface graphique.

> *« Donc, là-dessus, c'est voir un petit peu comment l'individu fonctionne : et un élément que je trouve intéressant là dedans, c'est de voir un petit leur document, comment ils font à l'heure actuelle. (...) Donc, ils sont déjà structurer selon leur vision à eux. On peut regarder et reprendre ça. Ce que l'on a fait en partie pour cette application»*

c) *Une démarche mixte pour les novices NSDK : des traitements dirigés par les concepts et par les données*

La démarche de résolution des novices se caractérise par un mode de fonctionnement dual. Deux raisons à cela :

1. Premièrement, *le poids des acquis professionnels* : le débutant reste encore très marqué par son expérience du site-central. Aussi, lorsqu'il est confronté à un problème qui présente une certaine ressemblance avec des situations passées, il se base sur ses représentations pour en extraire des caractéristiques qui lui rappellent les solutions qu'ils appliquaient auparavant. Croyant reconnaître certaines analogies, il déclenchera alors automatiquement les traitements correspondants. On se trouve alors dans le cas d'un traitement dirigé par les concepts. Cependant, comme nous l'avons montré dans l'analyse quantitative des solutions, la plupart des schémas récupérés se révéleront inappropriés à l'usage.

2. Deuxièmement, *les caractéristiques de la conception graphique* déterminent "*un traitement dirigé par les données*" : le novice agit de la même manière que l'expert, c'est-à-dire qu'il se base sur les spécifications du projet pour extraire les éléments qui pourront être directement exploités sous forme d'objets graphiques dans la conception. Mais à la différence de leurs "aînés", on remarque que ces novices éprouvent de très grandes difficultés à proposer des représentations graphiques pertinentes. Leur inexpérience dans ce genre de pratique les pénalise en effet.

C'est en particulier lors de la tâche de dénomination des objets de l'interface que cela a été observé. Cette opération consiste à affecter un titre aux menus, aux boutons et aux fenêtres de la maquette. Les libellés doivent être aussi proches que possible du vocabulaire métier de l'utilisateur..

« Normalement, ce sont des verbes que l'on met [sur les boutons, NDR]*: donc, il va falloir essayer de travailler là-dessus, trouver un verbe : affiner, détailler, améliorer. C'est toujours la difficulté de trouver des mots qui représentent quelque chose : demander aux utilisateurs éventuellement, c'est peut-être même ce que l'on va faire d'ailleurs »*

Cette tâche, simple en apparence, se révèle en réalité assez délicate pour ces novices qui n'ont jamais eu à effectuer de telles traitements sémantiques sur le site-central. En effet, le nom des commandes était proposé directement par Pacbase, ou bien récupéré des anciens projets. Alors qu'en client serveur, l'informaticien doit déterminer, seul, la dénomination des objets sélectionnés. Et c'est justement ce qui lui pose problème.

« En site-central, on le savait : F3, c'était pour sortir ; F4, le menu général. Il y a avait des normes que l'on connaissait, et là c'est ça qui nous manque. Ici, en graphique, si je mets "importer" et "transférer", l'utilisateur pourra vite se mélanger les crayons. C'est deux notions proches, mais qui ne reflètent pas les mêmes traitements. Il faut alors choisir le bon qualificatif. Mais lequel ? c'est tout le problème »

d) En résumé

i) il s'avère que la conception sur Pacbase implique plutôt un **mode de traitement dirigé par les concepts**. La bonne connaissance que les informaticiens ont des besoins des utilisateur ainsi que la relative homogénéité de ses demandes, leur permettent de reconnaître très vite le genre de problème à traiter et de proposer d'emblée une solution admissible.

ii) En revanche, la conception graphique exige plutôt un mode de fonctionnement **dirigé par les données**. La faculté de modéliser directement les entités du monde réel en objets graphiques favorise ce type de traitement.

iii) Enfin, le novice est partagé entre ces deux types de résolution : un traitement **dirigé par les données et par les concepts**. Le premier est requis pour concevoir une interface qui colle précisément à l'univers des utilisateurs. Le second est activé à cause des expériences passées trop prégnantes de l'informaticien. Cela dit, l'absence d'expériences significatives handicapent

également ces développeurs, notamment lorsqu'elles ne fournissent pas de supports de résolution suffisamment pertinents pour comprendre un problème grâce à un transfert d'apprentissage positif (cas de la dénomination des objets graphiques par exemple).

3.2.3.2.5 Processus d'évaluation et d'ajustement des solutions

On a indiqué, à travers les processus de résolution, quel rôle pouvait jouer l'environnement technique sur les stratégies de production de solution (par analogie *Vs* par construction). Nous avons également montré comment l'expérience de l'informaticien et les caractéristiques de la conception graphique pouvaient déclencher un mode de traitement particulier (dirigé par les concepts *Vs* par les données). Cette dernière partie est consacrée à l'analyse des stratégies d'évaluation et de sélection des solutions mises en oeuvres par :

a) les experts Pacbase ; avec une stratégie en "*largeur d'abord*"
b) les experts NSDK : avec une double stratégie en "*largeur d'abord*" et en "*profondeur d'abord*"
c) les novices NSDK : avec une stratégie en "*profondeur d'abord* "

a) Chez l'expert Pacbase : une évaluation en largeur d'abord

Il peut arriver que plusieurs projets identifiés dans la bibliothèque de Pacbase conviennent et offrent du même coup, plusieurs solutions admissibles pour un même problème. L'informaticien doit alors procéder à des choix. La stratégie employée est une stratégie en "*largeur d'abord*". Concrètement, cela consiste à tester plusieurs solutions simultanément et à s'arrêter dès qu'une solution de conception paraît meilleure qu'une autre. Ce processus de sélection n'est possible que parce que l'informaticien est expérimenté et qu'il maîtrise l'ensemble des solutions candidates. La sélection de ces solutions s'effectuera principalement sur la base de contraintes de conception.

"Ce que l'on regarde en premier, lorsqu'on va choisir un écran de maquettage, c'est qu'il ne consomme pas trop de CPU lors des compilations. Si ça dépense trop, on risque de se faire taper sur les doigts. Il faut donc en trouver une autre."

b) Chez l'expert NSDK : une évaluation en largeur et en profondeur

Dans cette situation de conception novatrice, l'informaticien expert est amené, avec l'aide du système, à élaborer un grand nombre de solutions au cours de la résolution ; solutions qu'il devra trier selon des critères précis.

Deux processus cognitifs sont dès lors susceptibles d'intervenir dans la sélection de ces solutions :

1. une stratégie en "*largeur d'abord*" qui est la même que celle décrite précédemment chez les experts Pacbase ;
2. une stratégie "*en profondeur d'abord*". Dans ce cas, l'informaticien déroule une seule solution à la fois en essayant d'estimer sa conformité par rapport à des critères de conception. Ces critères correspondent à ceux édictés par le service méthode. Ainsi, si cette solution ne remplit par les conditions fixées par la charte de développement (*nombre de boutons, de fenêtres, temps de réponse...*), l'informaticien l'abandonnera pour en tester une autre.

Le déclenchement d'une de ces deux stratégies d'évaluation dépendra de la familiarité et de la complexité des solutions manipulées :

- si elles sont complexes et originales ; l'informaticien préférera les tester une par une, par une évaluation "*en profondeur d'abord*".
- En revanche, si ces schémas de résolution lui sont familiers, une stratégie d'évaluation "*en largeur d'abord*" suffira.

c) Chez le novice NSDK : une évaluation en profondeur d'abord

La démarche d'évaluation employée par les novices repose sur une stratégie "*en largeur d'abord*". Dans le cas de problèmes réellement nouveaux (comme par exemple, la programmation des événements de la maquette), le novice peut très difficilement retrouver une solution admissible dans son répertoire d'expériences passées. Il lui faut donc construire une première solution susceptible de résoudre le problème rencontré. Il ne recherchera une solution alternative que si la première évoquée se révèle insatisfaisante lors de l'évaluation. Bien que cette stratégie soit la même que celle des experts, les raisons de son utilisation divergent :

- les concepteurs non expérimentés peuvent hésiter à s'engager dans une tâche qu'ils ne maîtrisent pas totalement. L'évaluation en "*profondeur*

d'abord" leur permet alors d'affiner la représentation des solutions et de mieux les maîtriser.

- Ils accordent aussi une confiance mitigée aux solutions qu'ils ont élaborées. Ce manque d'assurance les conduit à évaluer précisément ces solutions avant de les appliquer.
- Enfin, l'évaluation "*en profondeur d'abord*" s'apparente à une sorte d'apprentissage par l'action, car en déroulant entièrement la solution, en la testant et en identifiant ses incohérences (avec l'aide d'un expert), l'informaticien peut faire évoluer plus favorablement sa solution et progresser ainsi dans la conception.

> *« Tu ne savais pas a priori ce que tu voulais faire ou tu en avais une idée vague, et en réalisant cela se précise. C'est ni plus, ni moins que du test et de la correction : tu essais ça fonctionne, tu continues. Sinon t'arrêtes, tu modifies ou tu réessayes autre chose»*

d) En résumé

Différentes stratégies d'évaluation sont susceptibles d'être mises en œuvre par les informaticiens en fonction de leur maîtrise de la tâche informatique :

i. En développement site-central, l'expert peut être confronté à plusieurs solutions admissibles. Il les évaluera par une **stratégie en largeur d'abord** et ne retiendra que la plus satisfaisante.
ii. Dans le cadre de la conception client serveur, le novice évoquera avec difficulté une seule solution susceptible de résoudre le problème considéré. Il l'apprécie par une **stratégie en profondeur d'abord**, et ne recherche une solution alternative que si la première évoquée se révèle insatisfaisante.
iii. L'expert en conception client serveur est quant à lui tiraillé entre ces deux démarches :
    * d'une part, son expérience lui permet d'envisager plusieurs solutions susceptibles de convenir au problème rencontré. Il s'engage alors dans un activité d'évaluation comparative par une **stratégie en largeur d'abord** ;
    * d'autre part, les problèmes complexes et nouveaux exigent qu'il développe des solutions originales en accord avec les contraintes de conception fixées par le service. Dans ce cas, l'informaticien adopte plutôt une **évaluation en profondeur d'abord**.

3.2.4 Synthèse - Discussion

Les résultats obtenus nous conduisent à développer deux niveaux de discussion :

a) le premier portera sur le "*déterminisme cognitif*" des dispositifs de conception utilisés. En d'autres termes, dans quelle mesure les système affectent les raisonnements des informaticiens ?

b) Le second envisagera les difficultés rencontrées par les informaticiens novices dans leur activité de conception.

a) Relations entre les environnements techniques et les stratégies cognitives mises en oeuvre

Les analyses quantitatives et qualitatives des processus de résolution ont identifié un certain nombre de stratégies utilisées par les informaticiens pour la tâche de maquettage. D'après ces résultats, il semble que les processus cognitifs responsables de la construction, de la sélection et de la mise en œuvre des solutions, dépendent non seulement du niveau d'expertise (informaticiens experts ou novices) mais aussi du type d'expertise des informaticiens (connaissance d'un environnement site-central et/ou client serveur).

Ainsi, la comparaison du nombre de solutions exprimées a permis de déterminer le "rendement cognitif" des informaticiens sur chaque environnement. Le plus faible rendement ayant été trouvé chez les novices NSDK qui exprimaient un grand nombre de solutions insatisfaisantes. Les explications avancées furent d'une part, le manque de rigueur des critères et des contraintes de conception et, d'autre part, les transferts d'apprentissage négatifs réalisés entre les deux situations de conception.

L'étude qualitative des solutions a révélé de nombreuses divergences entre les stratégies mises en œuvre (Cf. Figure 65)

	Experts Pacbase	Experts NSDK	Novice NSDK	Déterminisme technique sur les stratégies cognitives?	Effets de l'expérience sur les stratégies cognitives ?
– ***Nature des problèmes à gérer***	Proche d'un problème de transformation d'états	Problème de conception	Problème de conception	Lié à l'environnement de conception (*environnement répétitif et prévisible Vs environnement nouveau et imprévisible*)	
– ***Représentation du problème***	Stable	Versatile	Versatile	Influence de l'environnement technique (contexte stable ou instable de la conception)	
– ***Nature des stratégies de résolution mises en œuvre***	Résolution analogique	Résolution dynamique et exploratoire	Résolution analogique, dynamique et par heuristique	Implication de l'environnement (dispositifs de simulation, stock de solutions proposées, contexte de conception innovant ou répétitif…)	Poids de l'expérience passée (surtout chez les novices : raisonnement analogique et théorie des schémas)
– ***Type de traitements mis en œuvre***	Dirigés par les concepts (Top-Down)	Dirigés par les données (Bottom-up)	Mixte des deux : Dirigés par les données et concepts	Influence de l'environnement technique (la conception par objet graphique favorise le traitement dirigé par les données)	Poids de l'expérience (les attentes des informaticiens déclenchent des modes de résolution)
– ***Processus de sélection / évaluation des solutions***	En largeur d'abord	En largeur d'abord, mais aussi En profondeur d'abord.	En profondeur d'abord		Selon l'expérience de l'informaticien, l'évaluation de la solution sera en largeur ou en profondeur d'abord.

Figure 65
Tableau récapitulatif des résultats du modèle de résolution

1. D'abord, les ***problèmes*** rencontrés en conception site-central se rapprochent davantage des problèmes de transformation d'état que de conception. Les problèmes de conception étaient quant à eux tout à fait caractéristiques des développements client serveur. Cette structure différenciée des problèmes était due aux caractéristiques des environnements qui assujettissaient (cas de Pacbase) ou au contraire affranchissaient (cas de NSDK) l'informaticien aux contraintes de conception. De plus, les réserves de solutions proposées par les sources externes (dictionnaire, librairie) influençaient également la nature des problèmes à traiter.

2. Par la suite, nous avons montré que la ***représentation du problème*** était différente selon l'environnement de conception dans laquelle elle était élaborée : plutôt statique en environnement site-central, plutôt versatile en environnement client serveur. Les marges de manoeuvre laissées par l'environnement, combinées à une gestion opportuniste de l'activité induisaient des représentations instables. A l'inverse, les trames de conception très rigides des squelettes de programmation de Pacbase ainsi que la relative homogénéité des problèmes rencontrés permettaient de produire une représentation qui restaient constante d'un bout à l'autre de la

conception. *Ici, l'environnement technique peut donc être tenu pour responsable du type de représentation élaborée.*

3. L'étude des ***stratégies de résolution*** a souligné trois types de démarche possible chez les informaticiens :
   i. une résolution *analogique* en conception site-central, qui consistait, sur la base d'une décomposition fonctionnelle du problème, à retrouver des solutions préexistantes dans le dictionnaire et à les appliquer.
   ii. une résolution *dynamique* des solutions chez les experts NSDK qui reposait sur la construction itérative des solutions par l'expérimentation et la simulation systématique des solutions élaborées.
   iii. et une *combinaison* de ces différentes stratégies chez les novices NSDK avec en plus, une démarche *heuristique* basée sur l'exploitation d'anciennes solutions. C'est selon le type de problème rencontré (familier ou non) et d'après la qualité et la quantité du stock de solutions disponibles, que l'informaticien allait privilégier l'une ou l'autre de ces différentes stratégies .

Ici, l'expérience de l'individu mais également les caractéristiques de l'environnement de conception semblent déterminer la mise en œuvre de ces conduites cognitives.

En effet, le contexte répétitif de *la conception site-central* fournit un large répertoire de réponses admissibles pour un problème posé. D'où la réutilisation facilitée des solutions. A l'opposé, l'absence de projets préexistants *en client serveur* oblige les informaticiens experts et novices à construire des solutions réellement innovantes. De même que les exigences des utilisateurs les poussent à expérimenter des modèles de maquettes toujours plus évolués.

Toutefois, *l'expérience de l'informaticien* joue un rôle tout aussi prépondérant dans le déclenchement de ces conduites cognitives dans la mesure où, comme on l'a vu chez le novice, le sujet puise des solutions dans son capital d'expériences pour les appliquer telles quelles ou pour en recomposer de nouvelles. Cela étant, la réutilisation de ces connaissances est souvent inadaptée au nouveau contexte de conception.

4. Une autre analyse a cherché à partir de quels stimuli ***les traitements cognitifs pouvaient être déclenchés*** :
    i. Dans le contexte de développement site-central, les traitements étaient *dirigés par les concepts* ; c'est-à-dire que la stabilité des situations de conception permettait à l'informaticien d'identifier rapidement une solution réutilisable, extraite des sources internes ou externes.
    ii. Dans le cadre de la conception client serveur, l'expert mettait plutôt en œuvre un traitement *dirigé par les données*. C'est la spécificité du mode de conception graphique, permettant de reproduire directement à l'écran les entités du monde réel, qui favorisait ce mode de fonctionnement cognitif.
    iii. Le novice utilisait alternativement *ces deux types de traitement* : soit *un traitement dirigé par les données* lorsqu'il percevait la possibilité de répliquer directement les données du problème sous forme de représentations graphiques ; soit *un traitement dirigé par les concepts*, lorsqu'il reconnaissait un problème familier qui provoquait le rappel d'une solution passée du site-central.

Ici, il semble donc bien que l'expérience de l'informaticien, mais de surcroît les caractéristiques propres à la conception graphique client serveur soient responsables de ces différents traitements.

5. Enfin, une dernière étude a déterminé ***les stratégies d'évaluation et de sélection de solution*** :
    i. En conception site-central, l'évaluation se faisait par une *stratégie en largeur d'abord* (comparaison simultanée de plusieurs solutions).
    ii. En conception client serveur, le novice exécutait une évaluation en *profondeur d'abord* : il développait une seule solution (en l'expérimentant) et la croisait avec des contraintes de conception pour juger de sa validité.
    iii. L'expert NSDK était *partagé entre ces deux modes de raisonnement*. Le choix dépendait de la complexité et de la nouveauté des solutions à appliquer.

Dans ce dernier cas, c'est davantage l'expérience de l'informaticien que l'environnement de conception qui paraît être responsable de la mise en œuvre de ces différentes stratégies d'évaluation.

b) Les erreurs et les difficultés des informaticiens novices

De nouvelles formes de conception ont donc fait leur apparition avec l'arrivée des outils de conception graphique ; en particulier l'émergence des démarches de conception événementielle et graphique dans un cadre de résolution original. Ces logiques supposent la remise en cause des raisonnements que l'informaticien avait l'habitude d'appliquer dans le contexte site-central. Il doit alors transformer une part non négligeable de ses anciens automatismes de résolution pour les adapter aux exigences du nouvel environnement.

Mais, pressé par les obligations de résultat, l'informaticien est très vite tenté de réutiliser les stratégies intellectuelles qu'il connaît le mieux ; à savoir celles issues du site-central. La plupart de ces analogies se révèlent toutefois incompatibles avec le cadre particulier de la conception graphique. En d'autres termes, les informaticiens se raccrochent à des anachronismes en conservant des pratiques qui n'ont plus aucun fondement en client serveur. Elles ne se justifient que par leur ancrage cognitif, héritage d'une période révolue. Aussi, les nombreuses erreurs de maquettage commises par les novices découlent de ces raisonnements obsolètes et de transferts d'apprentissage négatifs. A titre d'exemples, on rappellera :

- des comportements incohérents de la maquette : l'informaticien procède à une programmation locale des événements, et non contextuelle ;
- l'enchaînement arbitraire des fenêtres : l'enchaînement des fenêtres suit plus souvent la logique séquentielle et synchrone de Pacbase, que celle événementielle et asynchrone prônée dans la conception graphique ;
- le choix de termes ambiguës pour désigner les représentations graphiques : les titres donnés aux objets graphiques (boutons, menus) sont peu significatifs ;
- la mauvaise répartition des commandes sur la fenêtre : leur répartition se base sur celle des écrans de Pacbase. On trouve ainsi un maximum d'informations concentrées sur une seule zone de l'écran (fenêtre ou menu).

Néanmoins, des transferts positifs semblent également s'opérer. Les stratégies de résolution développées dans NSDK sont alors l'extension réussie de procédures extraites de l'environnement Pacbase. En particulier, on citera le cas de la démarche de composition graphique des fenêtres qui s'inspire de la technique de maquettage dynamique de Pacbase.

Après avoir abordé les démarches de résolution en œuvre dans chaque environnement informatique, nous allons tâcher de mettre en évidence les modèles d'interaction relatifs à chaque dispositif de conception.

3.3 MODÈLE D'INTERACTION

3.3.1 Méthode employée

a) Rappel de l'hypothèse

L'objectif de cette partie est de montrer que les modalités de fabrication des écrans (par codage sur Pacbase et par manipulation directe sur NSDK) conditionnent à la fois la qualité de la coopération homme-machine et l'implication du sujet dans la conception (engagement *Vs* retrait). L'hypothèse est que cette collaboration se révélera plus intense avec l'éditeur graphique NSDK qu'avec la grille de maquettage de Pacbase, parce que l'interface graphique rapproche les univers conceptuels de l'informaticien et de la machine.

b) Techniques de recueil utilisées

Pour évaluer la qualité de la coopération homme-machine et étudier l'engagement de l'informaticien dans la conception, nous nous sommes basés sur les corpus recueillis lors des verbalisations simultanées à l'activité. Notre attention s'est portée plus particulièrement sur la manière dont l'informaticien collaborait avec le système : quelles étaient les modalités de l'interaction homme-machine, quelles étaient les difficultés rencontrées, comment s'y prenait-il pour naviguer dans le système...

Par ailleurs, nous avons observé des situations d'interaction homme-machine en cherchant à déterminer les éventuels obstacles à cette collaboration, en particulier lors de la composition des fenêtres et durant la navigation sur les dispositifs techniques.
Notons que les caractéristiques de ces différents recueils (nombre d'informaticiens, durée et cadre des enregistrements...) sont les mêmes que celles que nous avions déjà définies pour les analyses sur les modèles d'action et de résolution.

Enfin, nous nous sommes livrés à une analyse ergonomique et fonctionnelle des outils de maquettage à partir d'une grille[55] d'évaluation développée sur la base des critères ergonomiques de Bastien et Scapin (1993).

[55] Cf. Annexe IV

c) Démarche d'analyse

A partir des données obtenues par verbalisation provoquée, deux types d'indicateurs ont été retenus :

1. Le premier indicateur s'inspire du modèle de Nanard (1991). Il évalue le niveau de coopération homme-machine selon les distances qui séparent l'univers mental de l'informaticien de l'univers physique du système informatique. Ainsi, plus ces distances sont ténues, meilleure sera la collaboration homme-machine. Pour cette analyse, trois distances ont été privilégiées :
   i. *Une distance sémantique* qui traduit l'effort mental que l'usager (ici l'informaticien) doit consentir pour assurer la transformation de sa solution conceptuelle en solution technique.
   ii. *Une distance articulatoire* qui traduit l'effort que l'individu doit réaliser pour mettre en correspondance les concepts du système et les représentations qui leur sont associées. L'usager est par exemple conduit à remplacer une séquence d'actions qui s'avérait pénible par une seule opération conceptuellement simple.
   iii. *une distance opératoire* qui traduit l'effort psycho-moteur que l'usager doit produire pour exprimer une commande ou interpréter un résultat.

2. Le second indicateur concerne la charge de travail occasionnée par les contraintes de chaque interface de conception. Nous pensons en effet que les efforts supplémentaires déployés par l'informaticien pour gérer ces contraintes affectent l'interaction homme-machine, et entravent du même coup le processus de rapprochement homme-machine. Pour identifier ces contraintes, nous nous sommes référés aux corpus de nos différents recueils (verbalisations + observations), ainsi qu'à l'inventaire des exigences qui avait été réalisé lors de la description du système homme-machine (Cf. Partie "*Description du chapitre homme-machine : l'analyse des contraintes*").

3.3.2 Présentation des résultats

Ce tableau rappelle les différentes modalités d'interaction proposées par les environnements de conception et précise la nature des distances relevées entre l'informaticien de son dispositif. Les vecteurs possibles de charge de travail sont également mentionnés (Cf. Figure 66).

	ENVIRONNEMENT PACBASE	ENVIRONNEMENT NSDK
Caractéristiques techniques		
Modalité de conception et d'interaction	codification sur grille de maquettage alphanumérique : saisie de codes	Manipulation directe sur interface graphique : manipulation de souris
Caractéristiques cognitives et opératoires de l'interaction homme-machine		
Distance sémantique	Forte	Faible
Distance articulatoire	Forte	Faible
Distance opératoire	Forte	Faible
Eléments générateurs de charge de travail	– Nombre de commandes importantes (près de 300) – Conception de la maquette sans visibilité – Mettre en accord sa représentation de la solution avec les termes techniques du système	– Arrêt intempestif du système – Normalisation de la conception (charte graphique)

Figure 66
Résultats du modèle d'interaction

D'une manière générale, on observe que les distances homme-machine sont les plus grandes dans le cadre de la conception Pacbase. En revanche, les modalités graphiques de l'interaction avec NSDK contribuent à une meilleure collaboration entre l'informaticien et son système, ainsi qu'à une plus grande spontanéité créative (possibilité de mettre directement en œuvre sa solution sans passer par le pont symbolique de l'interprétation). Mais nous avons aussi relevé que certaines contraintes (dysfonctionnement du système, normalisation de la conception) pouvaient interférer dans ce processus de rapprochement, et remettre en cause les bénéfices de la conception graphique, surtout chez le novice. Nous allons maintenant procéder à l'examen détaillé de ces différents résultats.

3.3.3 Analyse des résultats

3.3.3.1 L'agencement des écrans sur Pacbase : une construction par codage

Comme il nous a déjà été donné de le voir[56], il existe deux grandes techniques de construction des écrans sur Pacbase : un "*maquettage statique*" par l'intermédiaire des grilles de saisie (appelé "*-CE*") et un "*maquettage dynamique*" qui s'opère directement sur un écran vierge de l'écran ("-L").

a) Le maquettage statique sur Pacbase : une conception en aveugle et sans visibilité

Dans la plupart des cas observés, l'informaticien reprend directement des grilles d'écran disponibles dans le dictionnaire qu'il va modifier selon les spécifications du projet à réaliser. Le positionnement des rubriques (données, libellés, commentaires...) se fait par la saisie de nouvelles coordonnées sur la grille de maquettage. Deux procédures sont possibles :

- un *positionnement absolu* qui situe la rubrique par rapport aux limites de l'écran ;
- un *positionnement relatif* qui définit la rubrique par rapport à celle qui lui est immédiatement attenante.

Après chaque paramétrage de rubrique, l'informaticien est obligé de se livrer à une simulation de la maquette ("*-Sim*") pour visualiser l'écran et réajuster, le cas échéant, les rubriques qui seraient mal placées.

> *« C'est vraiment une conception qui est virtuelle : on l'imagine déjà et puis après on va voir et on affine petit à petit. C'est très contraignant. Si on s'aperçoit que cela n'est pas aligné, on est obligé de revenir (...) on revient, on modifie, on rajoute de 1. C'est pas mal contraignant. »*

Finalement, ce mode de fonctionnement accroît les écarts entre les représentations conceptuelles de l'informaticien et celles fonctionnelles du système, selon différentes modalités :

[56] Cf. description de l'activité de maquettage dans l'analyse du système homme-machine

1. *Distance articulatoire élevée*

L'informaticien n'a pas accès directement aux résultats de ses actions lors du maquettage. Il doit se représenter mentalement l'organisation de l'écran pour déterminer la disposition des rubriques *via* la grille de saisie. Parce qu'il n'a pas de visibilité sur ce qu'il produit, on dira que l'informaticien réalise cette tâche en "*aveugle*". Sa mémoire de travail est alors énormément sollicitée durant cette activité car il doit se rappeler de l'emplacement des rubriques à l'écran pour éviter les chevauchements lors du codage. Ces efforts de mémorisation contribuent à l'accroissement de sa charge mentale.

> *« ici, je fais un "-Sim" pour appeler l'écran et voir le résultat de ce que j'ai programmé. Je ne peux pas voir en direct. Cela m'oblige à imaginer l'écran et à concevoir quasiment en virtuel dans ma tête. C'est fatiguant à la fin ! (...)».*

2. *Distance sémantique élevée*

La tâche de maquettage, par grille de saisie interposée, implique que le développeur formule d'abord sa stratégie de résolution avant de procéder à son codage par un langage spécifique (instructions Pac). C'est une charge de travail supplémentaire si l'on considère que cette étape l'oblige à modifier la représentation de son action, en définissant de nouveaux buts et sous-buts compatibles avec le paradigme de fonctionnement de Pacbase. L'objectif étant de déterminer un mode opératoire qui sera adapté aux contraintes du dispositif.

> On a par exemple observé que le positionnement d'un rubrique en mode "relatif" ne pouvait se faire spontanément ; l'informaticien devait alterner diverses opérations :
> i. identifier la rubrique voisine qui servira de référence,
> ii. déterminer les coordonnées de cette rubrique;
> iii. évaluer à quelle distance il faut placer la nouvelle rubrique de celle déjà présente ;
> iv. traduire cette distance sous la forme de coordonnées.

C'est donc seulement après avoir effectué toutes ces opérations que l'informaticien pourra procéder au codage de la rubrique sur la grille de maquettage.

3. *Distance opératoire élevée*

Il existe un grand nombre de commandes (près de 300) qui peuvent être utilisées pour naviguer dans Pacbase et codifier les instructions de maquettage. La syntaxe et les règles d'écriture complexes[57] de ce langage sont responsables de nombreuses

[57] Par exemple, la commande "*E...XP.....P...*" est formée de plusieurs commandes signifiant "*Utilisations de la rubrique (code E)... dans la programme (code P) ... à partir des lignes P...,* "

erreurs, notamment lorsque l'informaticien confond un code avec un autre. Les développeurs avouaient d'ailleurs éprouver fréquemment des difficultés par se remémorer des commandes, après une interruption de quelques jours. En outre, les messages délivrés par le système doivent faire l'objet d'une réinterprétation par l'informaticien car ils sont délivrés dans un jargon technique, incompréhensible en l'état.

> *« là, il me dit que la rubrique que j'ai placée bouleverse l'écran. Pour vous, c'est certainement du charabia mais à force on s'y fait. Parfois, on oublie quand même ce que cela veut dire ».*

Aussi, pour ce manque de convivialité, la coopération avec Pacbase est loin d'être un processus naturel car l'informaticien doit constamment trouver la signification des codes pour comprendre cet environnement et interagir avec lui.

b) Le maquettage dynamique sur Pacbase : une conception visible

Ici, l'informaticien part d'un écran vierge et place directement les rubriques aux endroits souhaités par une codification particulière. Cette technique, beaucoup moins utilisée que la précédente, ne fait donc plus appel à la grille de maquettage. Toutefois, si l'informaticien ne conçoit plus en aveugle, il n'en reste pas moins qu'il reste confronté aux mêmes contraintes que le maquettage "statique" ; à savoir *(i)* l'obligation d'utiliser des codifications spécifiques pour mettre en œuvre sa solution de maquettage (distance sémantique) et *(ii)* de retenir des codes nombreux et difficilement mémorisables (distance opératoire élevée).

En définitive,

- la conception sans visibilité de la maquette (*obligeant l'informaticien à se représenter mentalement l'écran lors de sa définition*) ;
- les contraintes de codification (*imposant à l'informaticien d'exprimer d'abord sa procédure de résolution pour ensuite la mettre en conformité avec les contraintes techniques du dispositif*) ;
- et le nombre élevé de codes et de commandes à retenir…

…peut être tenu pour responsable d'une part, de l'accroissement des distances sémantiques, articulatoires et opératoires entre l'informaticien et son environnement et, d'autre part, de l'élévation de la charge mentale de ce développeur. Ce dernier est

en effet amené à mobiliser davantage de ressources cognitives pour faire face aux nombreuses exigences de son environnement.

Cette incompatibilité cognitive induit deux effets majeurs : fragilisation de la coopération homme-machine et réduction de l'engagement de l'informaticien dans la conception site-central. En effet, les efforts que celui-ci doit déployer pour accorder sa logique d'utilisation à la logique de fonctionnement du système ne favorisent pas une utilisation intuitive du dispositif. Bien au contraire, les ressources qu'il met en œuvre pour adapter ses conduites de travail l'épuisent mentalement et diminuent d'autant plus l'efficience de ses actions. Ne pouvant coopérer naturellement avec le système, il sera alors moins enclin à s'investir dans la conception pour trouver des solutions originales.

3.3.3.2 L'agencement des fenêtres avec NSDK : une conception par manipulation directe

a) Chez l'expert NSDK

Le maquettage sur NSDK se fait par manipulation directe. Concrètement, l'informaticien sélectionne les objets graphiques sur une palette d'icônes et les place directement sur les fenêtres à l'aide de la souris.

> *« Pour la construction, on a l'avantage d'avoir des outils qui sont nettement plus conviviaux. Puisque finalement, l'écran se constitue quasiment entièrement avec la souris ».*

1. *Réduction de la distance sémantique*

   La manipulation directe établit de nouvelles modalités de représentation et d'interaction avec l'environnement de maquettage. Elle réduit l'impression de distance sémantique que l'usager perçoit entre *i)* son intention, *ii)* son action et *iii)* le résultat de son action. En d'autres termes, l'informaticien n'a plus à traduire sa pensée par un codage complexe pour obtenir un comportement du système ; il agit directement sur l'interface pour "mettre en scène" ses intentions. De plus, l'interface lui donne en temps réel le résultat de ses actions ; il peut alors procéder à un réajustement dynamique de ses interventions par une boucle rétroactive.

   > *«C'est très agréable à faire, parce que l'on prend les objets, on les pose. Ça c'est super. On les déplace... tout ça avec la souris. Et puis on voit tout ce qui se passe au fur et à mesure de l'évolution, quand on teste : le test sur NSDK est très facile. Hop on a mis en place un écran qui enchaîne sur un autre, on voit tout de suite ce qui se passe. »*

2. *Réduction de la distance articulatoire*

En permettant un maximum de spontanéité créative, la conception graphique diminue la distance articulatoire. Concrètement, cela se traduit par la capacité à mettre en œuvre directement une hypothèse de conception et d'implémenter immédiatement les objets mentaux identifiés lors du processus de résolution. On retrouve là à nouveau l'idée de "*traitement dirigé par les données*" abordés dans le modèle de résolution.

> *« Je vais vouloir créer une combo BOX : je vais la prendre dans la barre d'outils et "Hop", et puis je l'intègre dans mon écran* [utilise la souris en même temps, NDR]. *Je la retaille, je lui change la police. Donc, là, on s'aperçoit que l'outil, pour créer le maquettage, c'est hyper plus convivial pour le développeur »*

3. Réduction de la distance opératoire

Alors que Pacbase exigeait la connaissance d'un grand nombre de commandes pour son utilisation, NSDK les propose directement sur des menus, des boîtes de dialogue et des boutons de commandes. De même, les instructions de maquettage sont remplacées par l'utilisation de la souris et par la manipulation directe. Les événements sont déclinés dans des boîtes de dialogue et tout un ensemble de fonctions réutilisables seront bientôt proposées par les librairies. La mémorisation des différents codifications techniques n'est donc pas aussi essentielle avec NSDK que cela pouvait l'être avec Pacbase. On peut considérer que ces différentes modalités d'interaction facilitent non seulement l'utilisation intuitive du système, mais favorisent aussi son apprentissage. Ce qui diminue d'autant plus la distance opératoire.

> *« Au niveau maquettage, l'outil est plus intuitif. J'en fais l'expérience régulièrement lorsque les nouveaux arrivent* [informaticien venant du site-central, NDR]. *Au bout de quelques instants, ils ont pigé le truc.»*

Un autre élément participe également au rapprochement homme-machine ; c'est l'appropriation du système par l'informaticien. En effet, ce dernier a la possibilité de personnaliser son environnement de conception (choix des couleurs, icônes, commandes, image de fond...) et d'y intégrer ses propres logiciels. D'une certaine manière, il recrée son propre espace de travail dans cet espace virtuel.

> *« J'ai l'impression d'avoir mon propre environnement. Je peux constituer mon poste de travail. Je peux personnaliser mon environnement et disposer de tous les avantages de ce type de machine »*

Par comparaison, rappelons que Pacbase imposait une interface standard et inamovible :

> *«On n'est vraiment dans notre monde, et on n'en bouge pas : on a notre bibliothèque et on n'en sort pas »*

Enfin, il est important de relever que tous les informaticiens insistent sur l'aspect ludique de la navigation et du maquettage avec NSDK. Les multiples médiateurs techniques proposés d'une part (manipulation directe, représentations iconiques, sélection de menus et de boutons...), et la construction de fenêtres graphiques animées d'autre part, contribuent à créer cette ambiance particulière de travail.

> *« Sur NSDK, je trouve que c'est plus sympa à utiliser et que cela permet de faire des choses beaucoup plus belles. C'est vrai qu'il y a un aspect ludique »*

Pour toutes ces raisons, la collaboration homme-machine semble donc plus intense dans l'environnement client serveur que dans le site-central. L'engagement de l'informaticien dans la conception graphique est également plus important à cause de cette proximité cognitive.

b) Chez les novices NSDK

Le très large engouement manifesté par les experts pour les modalités d'interaction de NSDK laissait présager un phénomène de rapprochement homme-machine similaire chez les novices, favorisant ainsi l'apprentissage et l'appropriation des nouveaux outils. Pourtant, les données recueillies révèlent une tout autre réalité. Les débutants sont en effet frustrés par les nombreuses règles de conception graphiques à respecter. Celles-ci proviennent des chartes édictées par la cellule méthode.

> *« Si tu veux vraiment créer une maquette rapide, cela peut être vraiment très rapide. En 5 minutes, tu crées une fenêtre, tu mets des contrôles dedans. Après, il y a les normes, c'est-à-dire à réajuster une position... Ca, c'est plus long. »*

En fait, les développeurs craignent de retrouver une situation de conception prescriptive comparable à celle du site-central ; à ceci près qu'elle ne se serait plus dirigée par les canevas techniques du système mais par les normes institutionnelles.

> *« (...) on est sceptique sur certains points : on veut une amélioration mais on ne veut pas revenir à l'âge de pierre de Pacbase avec des contraintes trop grosses qui limitent l'utilisation de l'environnement graphique»*

La manipulation directe exige également une habileté opératoire pour le positionnement et le dimensionnement des objets graphiques sur les fenêtres (faire bouger la souris par exemple). Seulement, ces modes d'action sont tout à fait inédits pour des novices qui n'ont jamais rien connu d'autre que le seul clavier du terminal comme dispositif d'interaction. Il y en a d'ailleurs qui ont de véritables difficultés pour composer les fenêtres avec la souris.

> *« C'est pas du tout évident de faire une fenêtre. On a tendance à la faire soit trop haute, soit trop large, soit trop grande (...). le maquettage est très long, je suis très étonné du temps parce qu'il faut compter, tout aligner, mettre la bonne police, mettre les bons accélérateurs... je trouve que c'est très très long à faire pour que cela soit vraiment parfait. Il y a un long travail de présentation. »*

Par conséquent, les nombreux tâtonnements auxquels se livrent ces développeurs pour placer correctement les objets vont avoir pour effet d'augmenter sa charge de travail. Ces difficultés opératoires compromettront d'autant plus le rapprochement homme-machine, et plus encore l'engagement de l'informaticien dans la conception.

Une dernière réserve formulée par ces débutants porte sur le manque de fiabilité de l'environnement client serveur. En effet, à tout moment, le système peut s'arrêter et supprimer tout travail non sauvegardé ("*plantage*"). A cause de ces nombreuses interruptions intempestives, l'informaticien a de moins en moins confiance dans le dispositif. Certains hésitent même à s'investir dans des tâches informatiques longues et fastidieuses qui peuvent s'annuler à tout moment. La fréquence de ces incidents techniques ainsi que leur régulation par l'informaticien est un facteur de charge de travail.

> *« Je dois rebouter mon PC dix fois par jour, alors que le terminal* [sur site-central, NDR] *tenait le coup. En fait Windows n'est pas un environnement parfait : il a beaucoup de défauts, et notamment, dans l'utilisation de NSDK. A la moindre petite erreur de programmation, on risque de tout faire sauter. On se demande alors pourquoi il plante, c'est énervant, c'est stressant »*

En somme, si NSDK offre un environnement graphique particulièrement convivial et ludique pour élaborer les maquettes, la normalisation de l'activité par les règles de conception d'une part, et l'instabilité quasi-chronique du système d'autre part, sont autant de facteurs qui nuisent à la coopération homme-machine et qui menacent, du même coup, l'investissement de l'informaticien dans la conception.

3.3.4 Synthèse-Discussion

Selon le type d'environnement utilisé, nous avons identifié différents niveaux de coopération homme-machine possibles conduisant à l'engagement de l'informaticien dans la conception. Ces attitudes sont représentées sur le graphique suivant (Cf. Figure 67) où apparaissent :

a) en **abscisse**, les deux niveaux d'engagement possibles de l'informaticien dans la conception : implication *Vs* retrait.

b) en **ordonnée**, les deux logiques d'action possibles :

- Soit une logique fonctionnelle imposée par le système :

  "Faire telle(s) opération(s) ou taper telle(s) instruction(s) → pour obtenir tel résultat".

  Elle correspond à la codification de l'interaction sur Pacbase ;

- Soit la logique d'utilisation de l'informaticien :

  "Pour obtenir tel résultat → faire telle(s) opération(s) ou taper telle(s) instruction(s)"

  Elle correspond à la manipulation directe de NSDK.

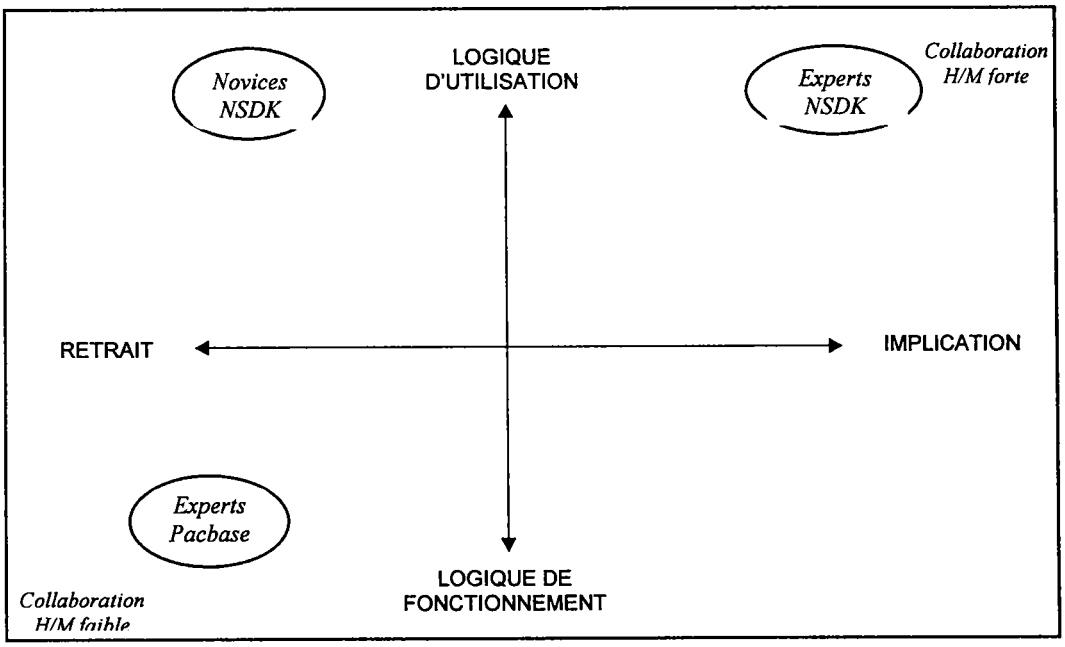


Figure 67
Coopération homme-machine et implication de l'informaticien dans le développement

Les *experts Pacbase* sont l'échantillon dont la coopération avec le système et l'engagement dans la conception sont les plus faibles. Les causes en sont *i)* le nombre élevé de commandes à retenir qui pénalise l'utilisation naturelle du système, *ii)* l'obligation qui est faite à l'informaticien de mettre en accord son modèle de pensée avec le paradigme de conception , et enfin *iii)* l'absence de visibilité du maquettage qui sollicite de manière trop excessive ses ressources cognitives (recours à de nombreuses anticipations et à la mémorisation).

Le coopération entre les *experts client serveur* et leur environnement de développement a été signalée comme plus profonde. Les raisons évoquées portent sur l'interface graphique qui, outre le fait qu'elle procure un aspect ludique à la conception, autorise un maximum de spontanéité créative par le biais de la manipulation directe. De plus, cet environnement requiert nettement moins de connaissances techniques que Pacbase pour être utilisé, et il a l'avantage d'être personnalisable.

Sur la base de ces constats, nous avons supposé que les *novices NSDK* allaient présenter des conduites similaires à l'égard du nouveau système et, notamment, que ce rapprochement pouvait favoriser l'appropriation de NSDK. Pourtant, les nombreuses critiques dont nous nous sommes fait l'écho ont montré la fragilité de cette coopération. L'instabilité chronique du système et le poids des règles de conception rebutent de plus en plus les informaticiens, et écornent sérieusement l'image de haute technicité de l'environnement.

Au total, si l'interface graphique permet d'ouvrir de nouvelles modalités de coopération, rapprochant les univers conceptuels du système homme-machine, cette proximité cognitive reste néanmoins précaire. Elle dépend de facteurs externes (chartes graphiques, dysfonctionnements techniques) ou internes à l'individu (habilités opératoires insuffisantes pour le maniement de la souris, par exemple).

3.4 MODÈLE DE RÉFÉRENCE

3.4.1 Méthode employée

a) Rappel de l'hypothèse

Nous abordons dans cette dernière partie les "*modèles de référence*" qui interviennent dans la composition des interfaces. Ce sont des représentations mentales fonctionnelles très générales auxquelles les informaticiens se réfèrent pour construire un programme ou une interface homme-machine. Elles peuvent aussi se définir comme des "*protoconcepts*", puisque les entités informatiques élaborées sont des exemplaires particuliers de ces modèles .

Nous supposons que ces modèles de référence sont étroitement associés aux environnements techniques. C'est pourquoi, changer d'environnement de conception implique également de transformer ces matrices mentales de programmation.

b) Techniques de recueil de données

Pour étudier ces modèles de référence, nous nous sommes servis des données recueillies lors des observations, des verbalisations simultanées[58], et par la technique du tri conceptuel. Pour cette dernière méthode, nous demandions aux informaticiens de citer et de classer les connaissances qu'ils estimaient indispensables dans leur activité de maquettage.

Ce recueil par tri-conceptuel s'est déroulé lors d'entretiens complémentaires ciblés, après les phases d'étude sur le terrain. Il a impliqué 3 informaticiens Pacbase, 3 experts de NSDK et 5 novices NSDK, sur une durée de 20 minutes environ. Précisons que ces entretiens correspondent en fait à ceux qui nous avaient déjà servis pour identifier les contraintes générées par les systèmes techniques (Cf. description du système homme-machine).

[58] Pour ces deux recueils de terrain (observation + verbalisation), on pourra retrouver les caractéristiques de leur passation dans l'analyse des trois modèles précédents.

c) Indicateurs retenus et méthode d'analyse

A partir du tri-conceptuel, nous avons tout d'abord inventorié toutes les connaissances formulées par les informaticiens. Puis nous avons cherché celles qui, par leurs caractéristiques et leurs propriétés intrinsèques, rentraient dans la composition de ces modèles de référence.

Ensuite, à partir des analyses "*in situs*", nous avons relevé tout ce qui pouvait indiquer que l'informaticien se référait explicitement à ces modèles de conception. Par exemple, les références aux comportements des utilisateurs finaux, l'élaboration de scénarii d'utilisation, les exemples pris sur d'autres logiciels, etc. Des extraits de corpus illustrent ces analyses.

3.4.2 Présentation des résultats

Le tableau ci-dessous répertorie l'ensemble des connaissances désignées par les informaticiens lors du tri-conceptuel (Cf. *Figure 68*). Elles se regroupent en :

1. *connaissances techniques* : c'est-à-dire l'ensemble des connaissances nécessaires pour *i)* naviguer dans l'environnement technique (site-central ou client serveur), *ii)* utiliser les outils de conception (Pacbase ou NSDK) et *iii)* programmer et maquetter (codes, instructions Pac, NCL, Cobol et C...) ;
2. *connaissances fonctionnelles* : ce sont toutes les connaissances concernant le domaine d'activité à informatiser (connaissances du métier, des utilisateurs, des spécifications fonctionnelles...)
3. *connaissances de présentation* : elles font référence aux connaissances requises pour organiser et structurer les écrans de la maquette.

	3 EXPERTS PACBASE	3 EXPERTS NSDK	5 NOVICES NSDK
CONNAISSANCES TECHNIQUES			
– Noms des commandes, des codes	*100%* [59] *(près de 300 à mémoriser)*	*33 %* [60] *(présentés sous forme graphique : boutons, menus, boîte de dialogue)*	*100% (ils doivent connaître dans quels menus, dans quelles boîtes de dialogue ces commandes se cachent)*
– Canevas de maquettage	*100 % (Squelettes de programmation, Grille de maquettage)*	Non Pertinent (NP)	NP
– Noms et fonctions des Objets, des Evénements…	NP	*100% (près d'une dizaine d'objets graphiques, plus d'une soixante d'événements)*	*100% (connaissances d'autant plus difficiles à retenir que les événements sont écrits en anglais)*
– Eléments réutilisables	*100% (Entités contenues dans les bibliothèques : écran, programmes, macro…)*	*100% (Fonctions réutilisables des libraires, mais peu nombreuses pour l'instant)*	*100% (Peu nombreuses pour l'instant)*
– Langages des outils de développement	*100% (Pac et Cobol)*	*100% (NCL et C)*	*100% (NCL et C)*
– Langages de requêtes des bases de données	*100% (DB2 et DL1)*	*100% (Sybase et procédures stockées)*	*100% (Sybase et procédures stockées)*
– Architecture technique	NP	*100% (définir les fonctions du client et du serveur : quel est celui qui gérera les traitements, les données, l'interface ?)*	*100% (définir les fonctions des postes client et serveur)*
– Logique de conception	*33% (1/3) (Logique transactionnelle imposée par le système)*	*100% (Logique événementielle programmée par l'informaticien)*	*100% (Logique événementielle programmée par l'informaticien)*
CONNAISSANCES FONCTIONNELLES			
– Sur le domaine d'activité à informatiser	*100%* Le domaine fonctionnel reste identique, quel que soit l'environnement technique	*100%*	*100%*
– Sur l'utilisateur final	*33% (facultatif)*	*100% (Importance extrême)*	*60% (3/5) (Importance extrême)*
– Sur les spécifications du projet	*100%*	*100%*	*100%*
CONNAISSANCES LIÉES AUX NORMES DE PRÉSENTATION			
– Sources d'inspiration	*100% (TSO + anciens projets)*	*100% (Environnement graphique Windows)*	*100% (Environnement graphique Windows)*
– Chartes graphiques de conception	NP (n'existe pas)	*100% (le respect de ces règles conditionne l'acceptation ou le refus de la maquette)*	*100% (le respect de ces règles conditionne l'acceptation ou le refus de la maquette)*

Figure 68
Tableau comparatif des différents types de connaissances requises lors de la conception

3.4.3 Analyse des Résultats

Dans un premier temps, nous détaillerons les différents types de connaissances formulés par chacune des trois catégories d'informaticiens. Dans un second temps, nous déterminerons les connaissances associées au modèle de référence et discuterons de leur fonction dans ce modèle.

[59] Dans ce cas, les trois informaticiens ont indiqué cette connaissance comme nécessaire
[60] Alors qu'ici, seul un informaticien sur trois a fourni cette réponse

3.4.3.1 Description des connaissances utilisées dans chaque environnement

a) Connaissances utilisées par les informaticiens experts Pacbase

Parmi les connaissances que ces experts jugeaient fondamentales pour mener à bien leur tâche de maquettage se trouvaient :

1) des connaissance techniques

- *Connaissance de l'ensemble des codes, des codifications, des libellés* pour commander, naviguer et programmer avec Pacbase
- *Connaissance des canevas* de programmation (squelettes TP et Batch) et de la grille de maquettage
- *Connaissance de la logique transactionnelle TSO* (module de traitement qui gère le dialogue de l'application : enchaînement des écrans)
- *Connaissance des entités disponibles sur les bibliothèques* : il faut savoir si elles existent, leur emplacement, les instructions pour les rapatrier ;
- *Connaissance des langages de Pacbase :* code Pac et "Cobol" généré ;
- *Connaissance des langages de requête de bases de données* : SQL pour gérer les bases DB2 et DL1 ;
- *Connaissance du paradigme de conception* : logique transactionnelle.

2) des connaissances de présentation

- *Connaissance des formats et des attributs des données à afficher* : par exemple, l'utilisation de la couleur rouge ou des majuscules pour représenter une rubrique répond à une fonction bien précise (attirer l'attention ou marquer le danger) ;
- *Connaissance des agencements particuliers des écrans* : l'organisation interne d'un écran dépend de sa vocation (un écran de consultation sera différent d'un écran de saisie).

3) des connaissances du domaine fonctionnel

- *Connaissance du domaine d'activité à modéliser et à représenter* : quel est le domaine d'activité à modéliser, quelles sont les fonctions caractéristiques à implémenter ? Y a-t-il des processus particuliers du métier à programmer (par exemple, des traitements statistiques, des conversions de devises...?)

- *Connaissance des niveaux de sécurité* : l'utilisateur est-il autorisé à faire telle ou telle transaction ?

b) Connaissances utilisées par les informaticiens experts NSDK

Bien que leur classement thématique soit analogue à celui des experts Pacbase, la nature des connaissances manipulées diverge :

1) Connaissances techniques

- *Connaissance de l'outil NSDK* : Il faut distinguer celles qui concernent la manipulation de l'outil (comme les commandes, les codes, les libellés, les menus, les options...) de celles qui relèvent des opérations de maquettage et de programmation (noms des objets graphiques, noms des événements, des fonctions de programmation, langage spécifique NCL, nom des librairies...).

  Remarquons aussi que la connaissance de ces entités de programmation dépasse le cadre de l'outil NSDK. L'informaticien s'informe également de tout ce qui se fait ou a été développé par les autres services, de manière à récupérer des fonctions ou des bouts de programmes déjà constitués.
- *Connaissance des logiciels bureautiques* utilisés pour des tâches additionnelles de développement : par exemple, le logiciel Word va servir pour la rédaction de la documentation de la maquette ; Excel pour des statistiques, Power-Point pour des présentations, etc.
- *Connaissance de l'architecture client serveur* : La répartition judicieuse des traitements, des données et de l'interface entre les postes "client" et "serveur" améliorera sensiblement les performances de l'application ;
- *Connaissances de Sybase* : quelles sont les procédures stockées utilisables ? Quelles sont les personnes à contacter pour effectuer des opérations sur les bases de données ?
- *Connaissances du paradigme de conception* : logique événementielle.

2) Connaissances fonctionnelles

- *Connaissance de l'utilisateur final* : quels sont ses besoins, ses habitudes, ses expériences, sa formation ?
- *Connaissance du domaine d'activité à modéliser et à représenter* : quel est le domaine d'activité à informatiser ? Quel est le vocabulaire utilisé ? Existe-t-il des processus spécifiques à programmer ?
- *Connaissance des spécifications détaillées du projet* : ce qui doit être fait, les plannings, les interlocuteurs, le budget, les ressources.

3) Connaissances touchant aux normes de conception (graphiques, logiques, etc.)

- *Connaissance des logiciels disponibles sur l'environnement Windows* à partir desquels le concepteur puise son inspiration pour élaborer les interfaces : connaître les logiciels existants, ce qu'ils font, le contenu et l'organisation de leurs fenêtres, ce qu'il est possible de reproduire dans les projets de développement, etc.
- *Connaissance de la charte graphique du service* : ce qui doit être fait et ce qu'il ne faut pas faire, les normes de présentation de l'interface, etc.
- *Connaissance des règles de programmation de la maquette* : par exemple, "*le code d'un événement devra toujours être déporté et centralisé dans un fichier de programme général à la fenêtre*".

c) Connaissances utilisées par les informaticiens novices NSDK

Les connaissances citées par les novices NSDK sont de même nature que celles évoquées plus haut par les experts. Inutile de les redétailler à nouveau. Ceci dit, il existe des divergences quant à l'intérêt que chaque échantillon accorde à ces connaissances dans la conception.

Ainsi, très peu d'experts NSDK estiment que la connaissance des "*commandes*" de l'outil est capitale pour la tâche de maquettage *(33% des experts)* alors que 100% des novices le pensent. Cette différence peut s'expliquer par le fait que les débutants considèrent que la maîtrise du système repose avant tout sur l'apprentissage de ses dispositifs de commandes. L'expert, en revanche, ne juge pas nécessaire de les retenir puisqu'elles sont facilement accessibles, *via* ses menus et ses boîtes de dialogue.

On note aussi que la connaissance des événements et des objets du maquettage se révèle fondamentale pour les informaticiens NSDK, toutes catégories confondues. Toutefois, si les experts indiquent vouloir se rappeler prioritairement des événements correspondants à des tâches de maquettage particulières, les novices axent plutôt leur apprentissage sur les fonctions que ces événements et les objets graphiques sont censés remplir dans l'interface. En effet, avec une dizaine d'objets graphiques et près d'une soixantaine d'événements disponibles, tous rédigés en anglais, le novice a parfois du mal à faire un tri et à déterminer ceux qui sont compatibles avec ses intentions de maquettage.

> *« Je reviens au choix des événements, car pour moi c'est loin d'être évident. Bon parce qu'il y a des événements qui se valent un petit peu (...) Choisir par exemple entre Get-focus et un autre... c'est pas évident. (...) En plus que c'est écrit en anglais, on n'a pas directement le sens de ce qu'ils font».*

Une autre différence notable porte sur la connaissance que les informaticiens accordent à l'utilisateur final et à son activité dans le processus de maquettage. Tous les experts NSDK mettent ainsi en avant la nécessité de connaître les pratiques de travail de cet opérateur pour concevoir une interface ergonomique (vocabulaire métier, enchaînement logique des opérations, automatisation de certaines tâches répétitives et astreignantes...). A l'inverse, seul un novice sur deux signale ce savoir comme un préalable nécessaire à l'intervention. En fait, ces concepteurs négligent l'utilisateur comme ils le faisaient déjà dans le site-central : les applications étaient alors réalisées en vase clos, sans jamais impliquer le principal intéressé. D'ailleurs, les résultats enregistrés chez les experts site-central confirment cette attitude, puisqu'ils ne sont que 33 % à rappeler le rôle fondamental de l'utilisateur dans le développement.

On remarque enfin que si 33 % des experts Pacbase estiment l'apprentissage du paradigme transactionnel comme nécessaire à la tâche de conception; la totalité des informaticiens NSDK (experts et novices confondus) considèrent la connaissance du paradigme événementiel comme un prérequis indispensable au maquettage. Ces chiffres peuvent s'expliquer par le contrôle qu'exercent les systèmes techniques sur les conduites du développeur. Dans le cas de site-central, l'informaticien ne peut déroger aux canevas de développement. La connaissance des différentes fonctions de programmation n'est pas nécessaire dans la mesure où elles défilent automatiquement à l'écran. De même, le paradigme transactionnel est inhérent à ces squelettes, et

s'impose donc à l'informaticien. Ce dernier ne peut que s'y soumettre, sauf si, comme nous l'ont confié quelques développeurs, il veulent contourner les prescriptions pour effectuer des modifications sur le code cobol généré.
A l'inverse, NSDK ne donne aucune indication pour concevoir la maquette. C'est à l'informaticien qu'échoit la responsabilité de sélectionner et de programmer les événements, en fonction du type de comportement qu'il veut faire adopter à l'interface. La maîtrise parfaite des arcanes de la programmation événementielle est alors une condition tout à fait fondamentale dans cette situation.

Ces différents classements nous conduisent à développer deux types de réflexion sur l'utilisation et le rôle de ces connaissances dans l'activité de l'informaticien.

a) La première concerne la "transmissibilité" des connaissances d'un environnement de conception vers un autre :
Certaines connaissances sont spécifiques à un environnement. C'est par exemple le cas des connaissances des canevas de conception de Pacbase que l'on ne retrouve pas dans NSDK. De même, l'administration de l'architecture client serveur requiert des savoirs originaux que l'architecture site-central n'est pas en mesure de fournir. Ici donc, la réutilisation de tels types de connaissances ne sert à rien.

Certaines connaissances paraissent, à première vue, similaires entre elles ; elles se distinguent néanmoins par leur structure et leurs propriétés. C'est par exemple le cas des paradigmes de conception (transactionnel vs événementiel) : tous deux spécifient une démarche de développement et déterminent la logique de fonctionnement de l'application. Cependant, transférer la logique transactionnelle dans le contexte client serveur conduit automatiquement à de mauvaises solutions de développement (comme cela a déjà été largement démontré dans le modèle de résolution).

Enfin, il existe des connaissances qui sont identiques aux deux environnements, et qui peuvent donc être réutilisées par les individus. Ce sont par exemple les connaissances portant sur le domaine fonctionnel à informatiser car, quel que soit le système de programmation employé, le métier de l'utilisateur reste toujours le même.

b) Le second axe de réflexion porte plus particulièrement sur les connaissances impliquées dans le modèle de référence. Trois catégories se détachent :

- *les connaissances liées au domaine fonctionnel* et plus spécifiquement à l'utilisateur : l'application est destinée à un individu qui va l'utiliser pour une tâche bien précise ;
- *des exemples d'application* (provenant de projets préexistants ou des logiciels grand public) qui vont servir de source d'inspiration pour la conception graphique des interfaces ;
- *des normes techniques* qui vont structurer le dialogue homme-machine.

Nous allons revenir sur ces différentes composantes et sur les rôles qu'elles peuvent jouer dans l'activité de maquettage.

3.4.3.2 Modèle de référence utilisé sur Pacbase : stabilité des références

Lorsque les projets antérieurs ne recèlent pas de trames de conception suffisamment pertinentes pour élaborer la maquette, l'informaticien se rapporte alors aux configurations techniques standards du site-central pour retrouver des modèles d'organisation des écrans ou de structuration du dialogue. Ces modèles se réfèrent à des fonctions techniques de l'environnement centralisé, appelées "TSO" :

> *« Parce que je programme sous gros système* [site-central, NDR], *j'essaie toujours d'avoir en tête un éditeur que l'on a sur TSO, c'est-à-dire un environnement pour la gestion du gros système. Et j'essaie d'avoir la même structuration».*

3.4.3.3 Modèle de référence utilisé sur NSDK : instabilité des références

a) Chez les experts NSDK

Dans NSDK, le développeur comble l'absence de guides formels de conception en faisant appel à deux types de représentation. Il s'agit :

1. *d'un modèle de conception graphique* développé à partir de l'environnement Windows. L'informaticien se base sur des applications bureautiques classiques pour trouver des idées et concevoir ses maquettes. Par exemple, il peut s'inspirer de l'organisation et du contenu des menus de Word pour déterminer les commandes de la maquette.

   > *« Quand je développe en NSDK, j'essaie d'avoir Excel ou Word en tête et de voir comment les menus sont organisés en fichier, édition... »*

2. *d'un modèle d'animation logique de l'interface* (c'est-à-dire de construction du dialogue homme-machine) basé sur des "scénarii d'utilisation probables" de l'application par l'utilisateur final.

> *«J'ai des actions là-dessus : "lier"; etc... Donc, on va partir de ça et l'idée que l'on a eu, c'est de se dire quand l'utilisateur va ouvrir son document, il va choisir s'il le veut simplement en visu : notion de "profil", de " droit", de type d'utilisateur qui sont en face... ou s'il veut en répartition. Donc, on va avoir ça : "OK", "Annuler" etc. Mais pour en arriver là, on s'est dit, on va avoir une fenêtre où on va répartir les données, et une fenêtre où on va les visualiser. »*

Ces modèles de référence ne sont pas figés ; ils s'enrichissent au gré de découvertes de l'informaticien. Il suffit par exemple qu'il trouve un nouvelle façon de représenter des fichiers sur la dernière version d'un progiciel pour que son modèle de référence évolue subséquemment.

> *« Une interface graphique, on peut la penser à un instant "T", puis dans la prochaine version elle peut très bien évoluer car on peut avoir mûri nous aussi, avoir découvert de nouvelles formes de présentation. Donc, une application de gestion, c'est trouver au début la meilleure solution par rapport à tout ce que l'on a pu voir, imaginer... mais c'est savoir évoluer aussi. »*

b) Chez les novices NSDK

Les observations effectuées sur le novice donnent des résultats assez intéressants quant à la manière dont ces modèles de référence évoluent, suite à la migration technique. En particulier, l'informaticien ne s'est pas totalement séparé de ses anciennes connaissances. Bien au contraire, il préexiste des "résidus" de ces représentations fonctionnelles dans celles élaborées pour la conception graphique. Ces recouvrements sont révélés par les nombreuses erreurs commises par l'informaticien lors de la construction des maquettes.

> *« J'ai tendance à faire des fenêtres* [sur NSDK, NDR] *qui ressemblent à des écrans* [Pacbase, NDR] *; c'est-à-dire très simple. Il y a un libellé de champ et puis il y a un champ que l'on peut saisir. (...) Alors que l'on peut enrichir par plein de gadgets et d'icônes.»*

Par ailleurs, le débutant éprouve également d'énormes difficultés pour proposer des métaphores graphiques pertinentes et originales, sous forme d'icônes notamment. Ici, le déficit de représentations fonctionnelles ainsi que le manque de pratique à la conception graphique semblent davantage à l'origine de ces difficultés qu'un transfert négatif d'apprentissage.

> *«On nous demande de faire des interfaces avec des icônes, des boutons, d'imaginer des fenêtres graphiques conviviales. Le problème, c'est que c'est la première fois qu'on nous demande de faire ça. Et je sais pas comment faire, où chercher les modèles ? »*

Finalement, les obstacles de la conception graphique sont de deux ordres: soit c'est l'empreinte de l'ancien modèle de référence qui est trop prégnante ; soit c'est l'absence totale d'expérience graphique qui ne permet pas à l'informaticien d'imaginer des présentations originales. C'est pourquoi, certains développeurs vont chercher à enrichir leur modèle de référence en prenant comme source d'inspiration les interfaces des applications bureautiques. On retrouve là une conduite qui avait déjà été décrite chez les experts :

> *« J'essaie de me documenter un maximum en lisant des "doc" techniques envoyées par les éditeurs de logiciels ou même en utilisant des logiciels sur l'ordinateur de mes gosses. Comme ça, je vois un peu ce qui se fait comme interface, et cela me donne des idées »*

De la même façon, les informaticiens se projettent en utilisateur final pour construire et évaluer les maquettes. Seulement, la logique fonctionnelle à laquelle ils se réfèrent n'a rien à voir avec la logique d'utilisation de l'opérateur. Si bien que les interfaces conçues ont encore moins de chances de respecter les règles de programmation événementielle client serveur.

> *« Comme je ne connaissais pas le client serveur, je me suis plus placé du côté de l'utilisateur, c'est-à-dire que si je trouve que ça* [un bouton, NDR], *c'est pas très parlant, je vais m'en apercevoir, parce que justement je ne suis pas habitué, je suis l'utilisateur de base, à la limite assez naïf ».*

3.4.4 Synthèse - Discussion

On a donc vu qu'il existait deux modèles de référence spécifiques à chaque environnement.

Dans l'environnement site-central, un modèle de référence statique qui est mobilisé lorsque le dictionnaire de Pacbase ne propose aucun écran réutilisable. L'informaticien se base alors sur un modèle technique pour élaborer l'organisation des écrans et leur cinématique d'enchaînement. Par rapport aux connaissances citées, ce modèle se fonde sur les normes techniques de l'environnement (*mode TSO, logique transactionnelle*) ainsi que sur des principes de structuration des écrans (*écrans de maquettage*).

Le modèle de référence des experts NSDK est un modèle plus évolutif qui repose sur deux représentations prototypiques : d'une part, un modèle graphique de type "Windows" pour le design de l'interface et, d'autre part, un modèle logique construit à partir de *"scénarii d'utilisation de l'interface"* pour animer dynamiquement l'interface. Ce modèle n'est pas statique comme le précédent puisqu'il s'enrichit au gré des découvertes sur le système et sur d'autres dispositifs informatiques.

On a montré enfin que le modèle de référence des novices NSDK se situait à l'intersection de ces deux modèles (Pacbase et NSDK) : il persistait des traces de l'ancien modèle dans celui élaboré pour NSDK. C'est d'ailleurs pour cette raison que l'informaticien commet des erreurs de maquettage, car il s'inspire de représentations obsolètes qui n'ont plus aucun fondement dans le nouveau contexte de conception. Toutefois, l'absence totale d'expériences graphiques –qui auraient pu être réutilisées opportunément par un transfert d'expérience positif– peut également être considérée comme une cause possible des difficultés rencontrées par ces novices.

Au final, il apparaît que la conception graphique requiert non seulement la rupture avec d'anciens modèles de référence inadaptés, mais qu'elle nécessite aussi l'acquisition de représentations symboliques et fonctionnelles adéquates.

CHAPITRE IV

Discussion générale et Conclusion

1. DISCUSSION GÉNÉRALE

L'ensemble des résultats obtenus tendent à démontrer que les environnements techniques affectent non seulement les processus cognitifs mis en jeu lors de la conception, mais qu'ils influencent aussi les conditions dans lesquelles se déroulent les transferts d'apprentissage des informaticiens débutants.

Quatre niveaux de discussion vont être successivement développés dans cette partie :

1. Tout d'abord, nous procéderons au rappel des principales conclusions de nos études. Celles-ci seront confrontées aux hypothèses.
2. Ensuite, les difficultés rencontrées par l'informaticien novice dans sa tâche de conception seront examinées.
3. Puis, nous tenterons d'expliquer ces difficultés en caractérisant les processus mentaux en œuvre durant le transfert de compétences.
4. Enfin, une discussion sera engagée autour du rôle formateur que peut jouer l'environnement technique dans la structuration de ces raisonnements et plus généralement sur les pratiques professionnelles de l'informaticien.

1.1 LES MODÈLES MENTAUX EN PRÉSENCE

Dans l'ensemble, tous les résultats confirment nos hypothèses dans le sens où les modèles mentaux qui ont été identifiés sont spécifiques à chaque environnement de conception.

Ainsi, dans **le contexte de conception de type procédural** (outil Pacbase, architecture site-central), la stratégie de conception mise en œuvre par les experts est apparue fortement planifiée et linéaire (*résultat du modèle d'action*). L'informaticien définit très précisément la façon dont il va atteindre ses objectifs, il se donne une méthode. Il définit la suite des transformations nécessaires ainsi que les séquences d'action correspondantes (choix de rubriques, de commandes, de projets à réutiliser...). La décomposition de ce plan est essentiellement descendante et

privilégie une recherche de solutions en largeur d'abord. Ce modèle d'action est imposé par la structure normative des canevas de conception de Pacbase. Il y a très peu d'originalité dans les solutions développées ; l'informaticien applique directement des solutions dans le plan de conception, en se fondant sur son expérience (traitements dirigés par les concepts) et sur des ressources externes très riches (dictionnaire d'entités). Dans ces conditions, les raisonnements de conception s'apparentent dès lors à des processus "automatisés" et "analogiques" car ils font appel à des montages préétablis, stockés sur les sources internes ou externes de l'informaticien (*résultat du modèle de résolution*).

Dans le **contexte de conception de type graphique et événementiel** (NSDK et client serveur), la démarche de résolution **des experts** s'est révélée située (*résultat du modèle d'action*). Les possibilités techniques du langage d'une part (offrant une très grande latitude d'action), et la faculté de modéliser directement les entités du problème en objets informatiques d'autre part (favorisant les traitements dirigés par les données), favorisent les multiples réajustements opportunistes du plan de conception. On a noté aussi que ces informaticiens suivent une démarche exploratoire et heuristique en testant et en évaluant systématiquement les solutions envisagées (*résultat du modèle de résolution*).

L'étude du modèle d'interaction a révélé que les distances[61] entre les représentations de l'informaticien et de sa machine sont beaucoup plus ténues dans l'environnement client serveur que dans celui site-central. La compatibilité des modalités graphiques de conception (interface graphique, manipulation directe...) avec les modes de représentation et d'action de l'informaticien favorise le rapprochement homme-machine et accentue, de ce fait, l'implication de l'informaticien dans la conception.

Enfin, les résultats obtenus dans le cadre de l'analyse du modèle de référence ont permis de constater que les représentations impliquées dans la construction des interfaces NSDK provenaient de références externes à l'environnement technique. D'une part, l'informaticien se projetait en utilisateur final pour élaborer le dialogue homme-machine et, d'autre part, il s'inspirait de logiciels très répandus (Microsoft Word, Excel) pour concevoir l'organisation des fenêtres. En outre, ses représentations n'étaient pas figées, mais évoluaient au gré de ses découvertes graphiques.

Les particularités de ces différents modèles mentaux nous ont conduits à étudier les réactions des novices NSDK qui, en tant qu'anciens développeurs site-central, présentaient un "profil cognitif" que l'on peut considérer comme très proche des raisonnements des experts Pacbase ; mais qui, à cause de la migration technologique, étaient amenés à acquérir une configuration mentale semblable à celle des experts NSDK. Leurs nombreuses erreurs et difficultés de maquettage ont révélé la gageure d'une telle entreprise.

1.2 PRINCIPALES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES

Nous avons décelé que l'informaticien novice éprouvait un réel embarras pour s'approprier un **modèle de résolution** "*exploratoire*" permettant de trouver des solutions originales à des problèmes inédits. Ce novice préfère récupérer d'anciens schémas de résolution (le plus souvent obsolètes) plutôt que d'en créer de nouveaux. L'exemple le plus probant était la conception du dialogue homme-machine qui reposait sur un mode transactionnel et non événementiel comme cela aurait dû être le cas.

D'autres difficultés ont révélé que l'absence de canevas de conception sur NSDK avait déstabilisé ces novices (**modèle d'action**). Plus à l'aise dans la mise aux normes que dans la pure créativité, la perte de ces guides les a conduits à adopter deux attitudes assez contradictoires dans la gestion de l'activité : soit la mise en œuvre prudente et circonspecte d'un plan d'action s'opposant à toute déviation opportuniste ; soit, au contraire, une gestion désordonnée et quasi-anarchique du plan d'action aboutissant à la conception d'une application inutilisable de type "*usine à gaz*" pour reprendre l'expression d'un informaticien.

Par le **modèle d'interaction,** on a vu que les effets secondaires et bénéfiques attendus de l'utilisation de nouvelles modalités graphiques de conception *–en termes d'appropriation de l'environnement et d'engagement dans la conception–* étaient compromis à la fois par les normes de maquettage draconiennes imposées par le service méthode, et par le manque de fiabilité technique du système. Faisant de moins en moins confiance au dispositif, l'informaticien risquait de se détacher de son environnement et d'être ainsi moins impliqué dans son travail.

---

[61] Sémantiques, articulatoires, opératoires (Cf. modèle d'interaction)

Enfin, l'étude **du modèle de référence** a démontré la persistance de représentations prototypiques du site-central dans le modèle destiné à la conception client serveur. La composition déséquilibrée des fenêtres, ou encore l'enchaînement des fenêtres (en mode synchrone) en étaient les preuves les plus flagrantes. Mais nous avons souligné aussi que la carence d'expériences préalables en matière de conception graphique pouvait également être considérée comme un véritable handicap cognitif, dans la mesure où le novice était dans l'incapacité de s'appuyer sur de telles ressources conceptuelles pour développer les solutions graphiques requises.

Le tableau ci-dessous résume les principales caractéristiques des différents modèles mentaux identifiés et retrace les difficultés rencontrées par l'informaticien débutant lors de l'apprentissage du nouveau dispositif (Cf. Figure 69).

	Modèle d'organisation	**Modèle de résolution**	**Modèle de l'interaction**	**Modèle de référence**
Experts Pacbase	Organisation hiérarchisée de l'activité	Exploitation analogique des solutions Raisonnements automatisés	Coopération H/M faible Engagement faible dans la conception	Modèle de référence statique (dépendant de l'outil et centré sur l'outil)
Experts NSDK	Organisation opportuniste de l'activité	Construction itérative et exploratoire de la solution Raisonnements heuristiques	Coopération H/M forte Engagement fort dans la conception	Modèle de référence dynamique (indépendant de l'outil et tourné vers l'extérieur)
Novices NSDK	Difficultés liées à l'absence d'assistance de l'outil	Erreurs de conception dues à des transferts négatifs	Collaboration H/M menacée par des contraintes méthodologiques fortes et par l'instabilité technique du système	Recouvrement entre les 2 modèles

Figure 69
Rappels des principaux résultats de l'étude

Les conduites adoptées par ces novices nous amènent à développer une série de réflexions concernant d'une part, les différentes stratégies de transfert d'apprentissage employées durant la transition technique, et, d'autre part, le rôle que peuvent jouer les environnements techniques dans la gestion et la réutilisation des connaissances. Ces deux niveaux de discussion vont nous permettre d'établir la part de responsabilité que tiennent les variables internes (facteurs cognitifs) et externes (facteurs techniques) dans les problèmes rencontrés par les informaticiens durant leur migration.

1.3 DEUX MODÈLES EXPLICATIFS DES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES

1.3.1 Du rôle des mécanismes de transfert d'apprentissage dans la réutilisation de compétences erronées

Les résultats obtenus dans les cadres de la description du système homme-machine et de l'analyse des différents modèles mentaux laissent entrevoir plusieurs possibilités quant à la transmissibilité des compétences entre les environnements source et cible.

D'abord, le modèle de référence a clairement démontré qu'une réutilisation *totale et systématique* des représentations développées pour le site-central était impossible, tant les exigences graphiques et logiques de chaque contexte sont antagonistes.
Nonobstant, ces mêmes analyses ont également montré que des transferts de compétence positifs pouvaient se dérouler lorsque des analogies existaient entre les façons d'utiliser les outils ou de fabriquer la maquette. Les modèles de résolution et d'interaction ont ainsi mis en évidence que la technique de "*maquettage dynamique*" de Pacbase pouvait servir de support d'apprentissage aux novices, en particulier lorsqu'ils apprenaient à composer la fenêtre par manipulation directe sur NSDK. De même, les deux systèmes génèrent des langages qui ont le même paradigme procédural (Cobol et C); aussi, l'informaticien n'a pas à redévelopper un nouvel apprentissage pour comprendre le code de programmation.

Toutefois, il est apparu aussi que les nombreuses difficultés de conception découlaient d'une réutilisation inappropriée de compétences et/ou d'un défaut d'ajustement de celles-ci. Ayant repéré des similarités entre certaines caractéristiques techniques et fonctionnelles des deux environnements, et mue aussi par cette propension naturelle qu'a tout individu de dépenser le moins d'effort cognitif pour disposer du plus grand effet possible –"*Principe de l'économie cognitive*" de Wisser (1995)–, l'informaticien novice se livre alors à des raisonnements analogiques, le plus souvent inappropriés, qui le conduit à formuler des réponses inexactes aux problèmes rencontrés. A titre d'exemples, on citera la structuration des objets graphiques sur les fenêtres, la gestion cohérente des événements, la programmation de la logique de fonctionnement de la maquette, etc.

En définitive, il se dégage trois types de **compétences** sous-jacentes à ces différentes modalités d'apprentissage. Il peut s'agir de :

1. *Compétences spécifiques* : ce sont des compétences qui sont particulières à un environnement de conception donné. Ne correspondant plus aux normes de la nouvelle situation, elles deviennent désuètes et obsolètes. Du coup, l'informaticien doit non seulement se résoudre à les abandonner, mais aussi à en développer d'autres plus appropriées. Il se trouve dès lors dans une situation d'apprentissage.
2. *Compétences analogiques* : certaines caractéristiques des environnements source et cible étant identiques, les compétences qui leur sont associées peuvent être réutilisées telles quelles dans le nouveau contexte de conception (ce sont certaines stratégies du modèle de résolution).
3. *Compétences similaires* : à première vue, le sujet présuppose une similarité entre certains éléments des situations source et cible, et en déduit que les compétences qui sont utilisées pour les gérer sont elles aussi identiques. En fait, il existe des différences plus ou moins profondes entre ces deux contextes qui requièrent des adaptations sur les compétences transférées.

Durant le transfert d'apprentissage, ces différentes compétences vont être prises en charge par des **processus de régulation** (Cf. Figure 70). Selon le type de compétences disponibles dans le domaine source (analogiques, similaires ou spécifiques) et d'après la nature des compétences recherchées dans le domaine cible, quatre processus de régulation sont susceptibles d'être appliqués:

1. *Processus d'éradication de compétences spécifiques* : il s'agit d'écarter les connaissances qui ne sont plus d'aucune utilité dans la nouvelle situation de travail.
2. *Processus de création de compétences* : c'est le fait de développer de nouvelles compétences lorsque l'informaticien en manque pour traiter un problème original. C'est l'apprentissage de nouvelles connaissances.
3. *Processus d'ajustement et de maturation de connaissances similaires* : ce mécanisme doit permettre d'enrichir et/ou d'adapter les compétences aux exigences spécifiques du nouveau contexte de conception. Ce processus

3.4.3.1 Description des connaissances utilisées dans chaque environnement

a) Connaissances utilisées par les informaticiens experts Pacbase

Parmi les connaissances que ces experts jugeaient fondamentales pour mener à bien leur tâche de maquettage se trouvaient :

1) des connaissance techniques

- *Connaissance de l'ensemble des codes, des codifications, des libellés* pour commander, naviguer et programmer avec Pacbase
- *Connaissance des canevas* de programmation (squelettes TP et Batch) et de la grille de maquettage
- *Connaissance de la logique transactionnelle TSO* (module de traitement qui gère le dialogue de l'application : enchaînement des écrans)
- *Connaissance des entités disponibles sur les bibliothèques* : il faut savoir si elles existent, leur emplacement, les instructions pour les rapatrier ;
- *Connaissance des langages de Pacbase :* code Pac et "Cobol" généré ;
- *Connaissance des langages de requête de bases de données* : SQL pour gérer les bases DB2 et DL1 ;
- *Connaissance du paradigme de conception* : logique transactionnelle.

2) des connaissances de présentation

- *Connaissance des formats et des attributs des données à afficher* : par exemple, l'utilisation de la couleur rouge ou des majuscules pour représenter une rubrique répond à une fonction bien précise (attirer l'attention ou marquer le danger) ;
- *Connaissance des agencements particuliers des écrans* : l'organisation interne d'un écran dépend de sa vocation (un écran de consultation sera différent d'un écran de saisie).

3) des connaissances du domaine fonctionnel

- *Connaissance du domaine d'activité à modéliser et à représenter* : quel est le domaine d'activité à modéliser, quelles sont les fonctions caractéristiques à implémenter ? Y a-t-il des processus particuliers du métier à programmer (par exemple, des traitements statistiques, des conversions de devises...?)

- *Connaissance des niveaux de sécurité* : l'utilisateur est-il autorisé à faire telle ou telle transaction ?

b) Connaissances utilisées par les informaticiens experts NSDK

Bien que leur classement thématique soit analogue à celui des experts Pacbase, la nature des connaissances manipulées diverge :

1) Connaissances techniques

- *Connaissance de l'outil NSDK* : Il faut distinguer celles qui concernent la manipulation de l'outil (comme les commandes, les codes, les libellés, les menus, les options...) de celles qui relèvent des opérations de maquettage et de programmation (noms des objets graphiques, noms des événements, des fonctions de programmation, langage spécifique NCL, nom des librairies...).

  Remarquons aussi que la connaissance de ces entités de programmation dépasse le cadre de l'outil NSDK. L'informaticien s'informe également de tout ce qui se fait ou a été développé par les autres services, de manière à récupérer des fonctions ou des bouts de programmes déjà constitués.
- *Connaissance des logiciels bureautiques* utilisés pour des tâches additionnelles de développement : par exemple, le logiciel Word va servir pour la rédaction de la documentation de la maquette ; Excel pour des statistiques, Power-Point pour des présentations, etc.
- *Connaissance de l'architecture client serveur* : La répartition judicieuse des traitements, des données et de l'interface entre les postes "client" et "serveur" améliorera sensiblement les performances de l'application ;
- *Connaissances de Sybase* : quelles sont les procédures stockées utilisables ? Quelles sont les personnes à contacter pour effectuer des opérations sur les bases de données ?
- *Connaissances du paradigme de conception* : logique événementielle.

2) Connaissances fonctionnelles

- *Connaissance de l'utilisateur final* : quels sont ses besoins, ses habitudes, ses expériences, sa formation ?
- *Connaissance du domaine d'activité à modéliser et à représenter* : quel est le domaine d'activité à informatiser ? Quel est le vocabulaire utilisé ? Existe-t-il des processus spécifiques à programmer ?
- *Connaissance des spécifications détaillées du projet* : ce qui doit être fait, les plannings, les interlocuteurs, le budget, les ressources.

3) Connaissances touchant aux normes de conception (graphiques, logiques, etc.)

- *Connaissance des logiciels disponibles sur l'environnement Windows* à partir desquels le concepteur puise son inspiration pour élaborer les interfaces : connaître les logiciels existants, ce qu'ils font, le contenu et l'organisation de leurs fenêtres, ce qu'il est possible de reproduire dans les projets de développement, etc.
- *Connaissance de la charte graphique du service* : ce qui doit être fait et ce qu'il ne faut pas faire, les normes de présentation de l'interface, etc.
- *Connaissance des règles de programmation de la maquette* : par exemple, "*le code d'un événement devra toujours être déporté et centralisé dans un fichier de programme général à la fenêtre*".

c) Connaissances utilisées par les informaticiens novices NSDK

Les connaissances citées par les novices NSDK sont de même nature que celles évoquées plus haut par les experts. Inutile de les redétailler à nouveau. Ceci dit, il existe des divergences quant à l'intérêt que chaque échantillon accorde à ces connaissances dans la conception.

Ainsi, très peu d'experts NSDK estiment que la connaissance des "*commandes*" de l'outil est capitale pour la tâche de maquettage *(33% des experts)* alors que 100% des novices le pensent. Cette différence peut s'expliquer par le fait que les débutants considèrent que la maîtrise du système repose avant tout sur l'apprentissage de ses dispositifs de commandes. L'expert, en revanche, ne juge pas nécessaire de les retenir puisqu'elles sont facilement accessibles, *via* ses menus et ses boîtes de dialogue.

On note aussi que la connaissance des événements et des objets du maquettage se révèle fondamentale pour les informaticiens NSDK, toutes catégories confondues. Toutefois, si les experts indiquent vouloir se rappeler prioritairement des événements correspondants à des tâches de maquettage particulières, les novices axent plutôt leur apprentissage sur les fonctions que ces événements et les objets graphiques sont censés remplir dans l'interface. En effet, avec une dizaine d'objets graphiques et près d'une soixantaine d'événements disponibles, tous rédigés en anglais, le novice a parfois du mal à faire un tri et à déterminer ceux qui sont compatibles avec ses intentions de maquettage.

> *« Je reviens au choix des événements, car pour moi c'est loin d'être évident. Bon parce qu'il y a des événements qui se valent un petit peu (...) Choisir par exemple entre Get-focus et un autre... c'est pas évident. (...) En plus que c'est écrit en anglais, on n'a pas directement le sens de ce qu'ils font».*

Une autre différence notable porte sur la connaissance que les informaticiens accordent à l'utilisateur final et à son activité dans le processus de maquettage. Tous les experts NSDK mettent ainsi en avant la nécessité de connaître les pratiques de travail de cet opérateur pour concevoir une interface ergonomique (vocabulaire métier, enchaînement logique des opérations, automatisation de certaines tâches répétitives et astreignantes...). A l'inverse, seul un novice sur deux signale ce savoir comme un préalable nécessaire à l'intervention. En fait, ces concepteurs négligent l'utilisateur comme ils le faisaient déjà dans le site-central : les applications étaient alors réalisées en vase clos, sans jamais impliquer le principal intéressé. D'ailleurs, les résultats enregistrés chez les experts site-central confirment cette attitude, puisqu'ils ne sont que 33 % à rappeler le rôle fondamental de l'utilisateur dans le développement.

On remarque enfin que si 33 % des experts Pacbase estiment l'apprentissage du paradigme transactionnel comme nécessaire à la tâche de conception; la totalité des informaticiens NSDK (experts et novices confondus) considèrent la connaissance du paradigme événementiel comme un prérequis indispensable au maquettage. Ces chiffres peuvent s'expliquer par le contrôle qu'exercent les systèmes techniques sur les conduites du développeur. Dans le cas de site-central, l'informaticien ne peut déroger aux canevas de développement. La connaissance des différentes fonctions de programmation n'est pas nécessaire dans la mesure où elles défilent automatiquement à l'écran. De même, le paradigme transactionnel est inhérent à ces squelettes, et

s'impose donc à l'informaticien. Ce dernier ne peut que s'y soumettre, sauf si, comme nous l'ont confié quelques développeurs, il veulent contourner les prescriptions pour effectuer des modifications sur le code cobol généré.
A l'inverse, NSDK ne donne aucune indication pour concevoir la maquette. C'est à l'informaticien qu'échoit la responsabilité de sélectionner et de programmer les événements, en fonction du type de comportement qu'il veut faire adopter à l'interface. La maîtrise parfaite des arcanes de la programmation événementielle est alors une condition tout à fait fondamentale dans cette situation.

Ces différents classements nous conduisent à développer deux types de réflexion sur l'utilisation et le rôle de ces connaissances dans l'activité de l'informaticien.

a) La première concerne la "transmissibilité" des connaissances d'un environnement de conception vers un autre :
Certaines connaissances sont spécifiques à un environnement. C'est par exemple le cas des connaissances des canevas de conception de Pacbase que l'on ne retrouve pas dans NSDK. De même, l'administration de l'architecture client serveur requiert des savoirs originaux que l'architecture site-central n'est pas en mesure de fournir. Ici donc, la réutilisation de tels types de connaissances ne sert à rien.

Certaines connaissances paraissent, à première vue, similaires entre elles ; elles se distinguent néanmoins par leur structure et leurs propriétés. C'est par exemple le cas des paradigmes de conception (transactionnel vs événementiel) : tous deux spécifient une démarche de développement et déterminent la logique de fonctionnement de l'application. Cependant, transférer la logique transactionnelle dans le contexte client serveur conduit automatiquement à de mauvaises solutions de développement (comme cela a déjà été largement démontré dans le modèle de résolution).

Enfin, il existe des connaissances qui sont identiques aux deux environnements, et qui peuvent donc être réutilisées par les individus. Ce sont par exemple les connaissances portant sur le domaine fonctionnel à informatiser car, quel que soit le système de programmation employé, le métier de l'utilisateur reste toujours le même.

b) Le second axe de réflexion porte plus particulièrement sur les connaissances impliquées dans le modèle de référence. Trois catégories se détachent :

- *les connaissances liées au domaine fonctionnel* et plus spécifiquement à l'utilisateur : l'application est destinée à un individu qui va l'utiliser pour une tâche bien précise ;
- *des exemples d'application* (provenant de projets préexistants ou des logiciels grand public) qui vont servir de source d'inspiration pour la conception graphique des interfaces ;
- *des normes techniques* qui vont structurer le dialogue homme-machine.

Nous allons revenir sur ces différentes composantes et sur les rôles qu'elles peuvent jouer dans l'activité de maquettage.

3.4.3.2 Modèle de référence utilisé sur Pacbase : stabilité des références

Lorsque les projets antérieurs ne recèlent pas de trames de conception suffisamment pertinentes pour élaborer la maquette, l'informaticien se rapporte alors aux configurations techniques standards du site-central pour retrouver des modèles d'organisation des écrans ou de structuration du dialogue. Ces modèles se réfèrent à des fonctions techniques de l'environnement centralisé, appelées "TSO" :

> *« Parce que je programme sous gros système* [site-central, NDR], *j'essaie toujours d'avoir en tête un éditeur que l'on a sur TSO, c'est-à-dire un environnement pour la gestion du gros système. Et j'essaie d'avoir la même structuration».*

3.4.3.3 Modèle de référence utilisé sur NSDK : instabilité des références

a) Chez les experts NSDK

Dans NSDK, le développeur comble l'absence de guides formels de conception en faisant appel à deux types de représentation. Il s'agit :

1. *d'un modèle de conception graphique* développé à partir de l'environnement Windows. L'informaticien se base sur des applications bureautiques classiques pour trouver des idées et concevoir ses maquettes. Par exemple, il peut s'inspirer de l'organisation et du contenu des menus de Word pour déterminer les commandes de la maquette.

   > *« Quand je développe en NSDK, j'essaie d'avoir Excel ou Word en tête et de voir comment les menus sont organisés en fichier, édition... »*

2. *d'un modèle d'animation logique de l'interface* (c'est-à-dire de construction du dialogue homme-machine) basé sur des "scénarii d'utilisation probables" de l'application par l'utilisateur final.

> *«J'ai des actions là-dessus : "lier"; etc... Donc, on va partir de ça et l'idée que l'on a eu, c'est de se dire quand l'utilisateur va ouvrir son document, il va choisir s'il le veut simplement en visu : notion de "profil", de " droit", de type d'utilisateur qui sont en face... ou s'il veut en répartition. Donc, on va avoir ça : "OK", "Annuler" etc. Mais pour en arriver là, on s'est dit, on va avoir une fenêtre où on va répartir les données, et une fenêtre où on va les visualiser. »*

Ces modèles de référence ne sont pas figés ; ils s'enrichissent au gré de découvertes de l'informaticien. Il suffit par exemple qu'il trouve un nouvelle façon de représenter des fichiers sur la dernière version d'un progiciel pour que son modèle de référence évolue subséquemment.

> *« Une interface graphique, on peut la penser à un instant "T", puis dans la prochaine version elle peut très bien évoluer car on peut avoir mûri nous aussi, avoir découvert de nouvelles formes de présentation. Donc, une application de gestion, c'est trouver au début la meilleure solution par rapport à tout ce que l'on a pu voir, imaginer... mais c'est savoir évoluer aussi. »*

b) Chez les novices NSDK

Les observations effectuées sur le novice donnent des résultats assez intéressants quant à la manière dont ces modèles de référence évoluent, suite à la migration technique. En particulier, l'informaticien ne s'est pas totalement séparé de ses anciennes connaissances. Bien au contraire, il préexiste des "résidus" de ces représentations fonctionnelles dans celles élaborées pour la conception graphique. Ces recouvrements sont révélés par les nombreuses erreurs commises par l'informaticien lors de la construction des maquettes.

> *« J'ai tendance à faire des fenêtres* [sur NSDK, NDR] *qui ressemblent à des écrans* [Pacbase, NDR] *; c'est-à-dire très simple. Il y a un libellé de champ et puis il y a un champ que l'on peut saisir. (...) Alors que l'on peut enrichir par plein de gadgets et d'icônes.»*

Par ailleurs, le débutant éprouve également d'énormes difficultés pour proposer des métaphores graphiques pertinentes et originales, sous forme d'icônes notamment. Ici, le déficit de représentations fonctionnelles ainsi que le manque de pratique à la conception graphique semblent davantage à l'origine de ces difficultés qu'un transfert négatif d'apprentissage.

> *«On nous demande de faire des interfaces avec des icônes, des boutons, d'imaginer des fenêtres graphiques conviviales. Le problème, c'est que c'est la première fois qu'on nous demande de faire ça. Et je sais pas comment faire, où chercher les modèles ? »*

Finalement, les obstacles de la conception graphique sont de deux ordres: soit c'est l'empreinte de l'ancien modèle de référence qui est trop prégnante ; soit c'est l'absence totale d'expérience graphique qui ne permet pas à l'informaticien d'imaginer des présentations originales. C'est pourquoi, certains développeurs vont chercher à enrichir leur modèle de référence en prenant comme source d'inspiration les interfaces des applications bureautiques. On retrouve là une conduite qui avait déjà été décrite chez les experts :

> *« J'essaie de me documenter un maximum en lisant des "doc" techniques envoyées par les éditeurs de logiciels ou même en utilisant des logiciels sur l'ordinateur de mes gosses. Comme ça, je vois un peu ce qui se fait comme interface, et cela me donne des idées »*

De la même façon, les informaticiens se projettent en utilisateur final pour construire et évaluer les maquettes. Seulement, la logique fonctionnelle à laquelle ils se réfèrent n'a rien à voir avec la logique d'utilisation de l'opérateur. Si bien que les interfaces conçues ont encore moins de chances de respecter les règles de programmation événementielle client serveur.

> *« Comme je ne connaissais pas le client serveur, je me suis plus placé du côté de l'utilisateur, c'est-à-dire que si je trouve que ça* [un bouton, NDR], *c'est pas très parlant, je vais m'en apercevoir, parce que justement je ne suis pas habitué, je suis l'utilisateur de base, à la limite assez naïf ».*

3.4.4 Synthèse - Discussion

On a donc vu qu'il existait deux modèles de référence spécifiques à chaque environnement.

Dans l'environnement site-central, un modèle de référence statique qui est mobilisé lorsque le dictionnaire de Pacbase ne propose aucun écran réutilisable. L'informaticien se base alors sur un modèle technique pour élaborer l'organisation des écrans et leur cinématique d'enchaînement. Par rapport aux connaissances citées, ce modèle se fonde sur les normes techniques de l'environnement (*mode TSO, logique transactionnelle*) ainsi que sur des principes de structuration des écrans (*écrans de maquettage*).

Le modèle de référence des experts NSDK est un modèle plus évolutif qui repose sur deux représentations prototypiques : d'une part, un modèle graphique de type "Windows" pour le design de l'interface et, d'autre part, un modèle logique construit à partir de *"scénarii d'utilisation de l'interface"* pour animer dynamiquement l'interface. Ce modèle n'est pas statique comme le précédent puisqu'il s'enrichit au gré des découvertes sur le système et sur d'autres dispositifs informatiques.

On a montré enfin que le modèle de référence des novices NSDK se situait à l'intersection de ces deux modèles (Pacbase et NSDK) : il persistait des traces de l'ancien modèle dans celui élaboré pour NSDK. C'est d'ailleurs pour cette raison que l'informaticien commet des erreurs de maquettage, car il s'inspire de représentations obsolètes qui n'ont plus aucun fondement dans le nouveau contexte de conception. Toutefois, l'absence totale d'expériences graphiques –qui auraient pu être réutilisées opportunément par un transfert d'expérience positif– peut également être considérée comme une cause possible des difficultés rencontrées par ces novices.

Au final, il apparaît que la conception graphique requiert non seulement la rupture avec d'anciens modèles de référence inadaptés, mais qu'elle nécessite aussi l'acquisition de représentations symboliques et fonctionnelles adéquates.

CHAPITRE IV

Discussion générale et Conclusion

1. DISCUSSION GÉNÉRALE

L'ensemble des résultats obtenus tendent à démontrer que les environnements techniques affectent non seulement les processus cognitifs mis en jeu lors de la conception, mais qu'ils influencent aussi les conditions dans lesquelles se déroulent les transferts d'apprentissage des informaticiens débutants.

Quatre niveaux de discussion vont être successivement développés dans cette partie :

1. Tout d'abord, nous procéderons au rappel des principales conclusions de nos études. Celles-ci seront confrontées aux hypothèses.
2. Ensuite, les difficultés rencontrées par l'informaticien novice dans sa tâche de conception seront examinées.
3. Puis, nous tenterons d'expliquer ces difficultés en caractérisant les processus mentaux en œuvre durant le transfert de compétences.
4. Enfin, une discussion sera engagée autour du rôle formateur que peut jouer l'environnement technique dans la structuration de ces raisonnements et plus généralement sur les pratiques professionnelles de l'informaticien.

1.1 LES MODÈLES MENTAUX EN PRÉSENCE

Dans l'ensemble, tous les résultats confirment nos hypothèses dans le sens où les modèles mentaux qui ont été identifiés sont spécifiques à chaque environnement de conception.

Ainsi, dans **le contexte de conception de type procédural** (outil Pacbase, architecture site-central), la stratégie de conception mise en œuvre par les experts est apparue fortement planifiée et linéaire (*résultat du modèle d'action*). L'informaticien définit très précisément la façon dont il va atteindre ses objectifs, il se donne une méthode. Il définit la suite des transformations nécessaires ainsi que les séquences d'action correspondantes (choix de rubriques, de commandes, de projets à réutiliser...). La décomposition de ce plan est essentiellement descendante et

privilégie une recherche de solutions en largeur d'abord. Ce modèle d'action est imposé par la structure normative des canevas de conception de Pacbase. Il y a très peu d'originalité dans les solutions développées ; l'informaticien applique directement des solutions dans le plan de conception, en se fondant sur son expérience (traitements dirigés par les concepts) et sur des ressources externes très riches (dictionnaire d'entités). Dans ces conditions, les raisonnements de conception s'apparentent dès lors à des processus "automatisés" et "analogiques" car ils font appel à des montages préétablis, stockés sur les sources internes ou externes de l'informaticien (*résultat du modèle de résolution*).

Dans le **contexte de conception de type graphique et événementiel** (NSDK et client serveur), la démarche de résolution **des experts** s'est révélée située (*résultat du modèle d'action*). Les possibilités techniques du langage d'une part (offrant une très grande latitude d'action), et la faculté de modéliser directement les entités du problème en objets informatiques d'autre part (favorisant les traitements dirigés par les données), favorisent les multiples réajustements opportunistes du plan de conception. On a noté aussi que ces informaticiens suivent une démarche exploratoire et heuristique en testant et en évaluant systématiquement les solutions envisagées (*résultat du modèle de résolution*).

L'étude du modèle d'interaction a révélé que les distances[61] entre les représentations de l'informaticien et de sa machine sont beaucoup plus ténues dans l'environnement client serveur que dans celui site-central. La compatibilité des modalités graphiques de conception (interface graphique, manipulation directe...) avec les modes de représentation et d'action de l'informaticien favorise le rapprochement homme-machine et accentue, de ce fait, l'implication de l'informaticien dans la conception.

Enfin, les résultats obtenus dans le cadre de l'analyse du modèle de référence ont permis de constater que les représentations impliquées dans la construction des interfaces NSDK provenaient de références externes à l'environnement technique. D'une part, l'informaticien se projetait en utilisateur final pour élaborer le dialogue homme-machine et, d'autre part, il s'inspirait de logiciels très répandus (Microsoft Word, Excel) pour concevoir l'organisation des fenêtres. En outre, ses représentations n'étaient pas figées, mais évoluaient au gré de ses découvertes graphiques.

Les particularités de ces différents modèles mentaux nous ont conduits à étudier les réactions des novices NSDK qui, en tant qu'anciens développeurs site-central, présentaient un "profil cognitif" que l'on peut considérer comme très proche des raisonnements des experts Pacbase ; mais qui, à cause de la migration technologique, étaient amenés à acquérir une configuration mentale semblable à celle des experts NSDK. Leurs nombreuses erreurs et difficultés de maquettage ont révélé la gageure d'une telle entreprise.

1.2 PRINCIPALES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES

Nous avons décelé que l'informaticien novice éprouvait un réel embarras pour s'approprier un **modèle de résolution** "*exploratoire*" permettant de trouver des solutions originales à des problèmes inédits. Ce novice préfère récupérer d'anciens schémas de résolution (le plus souvent obsolètes) plutôt que d'en créer de nouveaux. L'exemple le plus probant était la conception du dialogue homme-machine qui reposait sur un mode transactionnel et non événementiel comme cela aurait dû être le cas.

D'autres difficultés ont révélé que l'absence de canevas de conception sur NSDK avait déstabilisé ces novices (**modèle d'action**). Plus à l'aise dans la mise aux normes que dans la pure créativité, la perte de ces guides les a conduits à adopter deux attitudes assez contradictoires dans la gestion de l'activité : soit la mise en œuvre prudente et circonspecte d'un plan d'action s'opposant à toute déviation opportuniste ; soit, au contraire, une gestion désordonnée et quasi-anarchique du plan d'action aboutissant à la conception d'une application inutilisable de type "*usine à gaz*" pour reprendre l'expression d'un informaticien.

Par le **modèle d'interaction,** on a vu que les effets secondaires et bénéfiques attendus de l'utilisation de nouvelles modalités graphiques de conception *–en termes d'appropriation de l'environnement et d'engagement dans la conception–* étaient compromis à la fois par les normes de maquettage draconiennes imposées par le service méthode, et par le manque de fiabilité technique du système. Faisant de moins en moins confiance au dispositif, l'informaticien risquait de se détacher de son environnement et d'être ainsi moins impliqué dans son travail.

---

[61] Sémantiques, articulatoires, opératoires (Cf. modèle d'interaction)

Enfin, l'étude **du modèle de référence** a démontré la persistance de représentations prototypiques du site-central dans le modèle destiné à la conception client serveur. La composition déséquilibrée des fenêtres, ou encore l'enchaînement des fenêtres (en mode synchrone) en étaient les preuves les plus flagrantes. Mais nous avons souligné aussi que la carence d'expériences préalables en matière de conception graphique pouvait également être considérée comme un véritable handicap cognitif, dans la mesure où le novice était dans l'incapacité de s'appuyer sur de telles ressources conceptuelles pour développer les solutions graphiques requises.

Le tableau ci-dessous résume les principales caractéristiques des différents modèles mentaux identifiés et retrace les difficultés rencontrées par l'informaticien débutant lors de l'apprentissage du nouveau dispositif (Cf. Figure 69).

	Modèle d'organisation	**Modèle de résolution**	**Modèle de l'interaction**	**Modèle de référence**
Experts Pacbase	Organisation hiérarchisée de l'activité	Exploitation analogique des solutions Raisonnements automatisés	Coopération H/M faible Engagement faible dans la conception	Modèle de référence statique (dépendant de l'outil et centré sur l'outil)
Experts NSDK	Organisation opportuniste de l'activité	Construction itérative et exploratoire de la solution Raisonnements heuristiques	Coopération H/M forte Engagement fort dans la conception	Modèle de référence dynamique (indépendant de l'outil et tourné vers l'extérieur)
Novices NSDK	Difficultés liées à l'absence d'assistance de l'outil	Erreurs de conception dues à des transferts négatifs	Collaboration H/M menacée par des contraintes méthodologiques fortes et par l'instabilité technique du système	Recouvrement entre les 2 modèles

Figure 69
Rappels des principaux résultats de l'étude

Les conduites adoptées par ces novices nous amènent à développer une série de réflexions concernant d'une part, les différentes stratégies de transfert d'apprentissage employées durant la transition technique, et, d'autre part, le rôle que peuvent jouer les environnements techniques dans la gestion et la réutilisation des connaissances. Ces deux niveaux de discussion vont nous permettre d'établir la part de responsabilité que tiennent les variables internes (facteurs cognitifs) et externes (facteurs techniques) dans les problèmes rencontrés par les informaticiens durant leur migration.

1.3 DEUX MODÈLES EXPLICATIFS DES DIFFICULTÉS RENCONTRÉES PAR LES NOVICES

1.3.1 Du rôle des mécanismes de transfert d'apprentissage dans la réutilisation de compétences erronées

Les résultats obtenus dans les cadres de la description du système homme-machine et de l'analyse des différents modèles mentaux laissent entrevoir plusieurs possibilités quant à la transmissibilité des compétences entre les environnements source et cible.

D'abord, le modèle de référence a clairement démontré qu'une réutilisation *totale et systématique* des représentations développées pour le site-central était impossible, tant les exigences graphiques et logiques de chaque contexte sont antagonistes.
Nonobstant, ces mêmes analyses ont également montré que des transferts de compétence positifs pouvaient se dérouler lorsque des analogies existaient entre les façons d'utiliser les outils ou de fabriquer la maquette. Les modèles de résolution et d'interaction ont ainsi mis en évidence que la technique de "*maquettage dynamique*" de Pacbase pouvait servir de support d'apprentissage aux novices, en particulier lorsqu'ils apprenaient à composer la fenêtre par manipulation directe sur NSDK. De même, les deux systèmes génèrent des langages qui ont le même paradigme procédural (Cobol et C); aussi, l'informaticien n'a pas à redévelopper un nouvel apprentissage pour comprendre le code de programmation.

Toutefois, il est apparu aussi que les nombreuses difficultés de conception découlaient d'une réutilisation inappropriée de compétences et/ou d'un défaut d'ajustement de celles-ci. Ayant repéré des similarités entre certaines caractéristiques techniques et fonctionnelles des deux environnements, et mue aussi par cette propension naturelle qu'a tout individu de dépenser le moins d'effort cognitif pour disposer du plus grand effet possible –"*Principe de l'économie cognitive*" de Wisser (1995)–, l'informaticien novice se livre alors à des raisonnements analogiques, le plus souvent inappropriés, qui le conduit à formuler des réponses inexactes aux problèmes rencontrés. A titre d'exemples, on citera la structuration des objets graphiques sur les fenêtres, la gestion cohérente des événements, la programmation de la logique de fonctionnement de la maquette, etc.

En définitive, il se dégage trois types de **compétences** sous-jacentes à ces différentes modalités d'apprentissage. Il peut s'agir de :

1. *Compétences spécifiques* : ce sont des compétences qui sont particulières à un environnement de conception donné. Ne correspondant plus aux normes de la nouvelle situation, elles deviennent désuètes et obsolètes. Du coup, l'informaticien doit non seulement se résoudre à les abandonner, mais aussi à en développer d'autres plus appropriées. Il se trouve dès lors dans une situation d'apprentissage.
2. *Compétences analogiques* : certaines caractéristiques des environnements source et cible étant identiques, les compétences qui leur sont associées peuvent être réutilisées telles quelles dans le nouveau contexte de conception (ce sont certaines stratégies du modèle de résolution).
3. *Compétences similaires* : à première vue, le sujet présuppose une similarité entre certains éléments des situations source et cible, et en déduit que les compétences qui sont utilisées pour les gérer sont elles aussi identiques. En fait, il existe des différences plus ou moins profondes entre ces deux contextes qui requièrent des adaptations sur les compétences transférées.

Durant le transfert d'apprentissage, ces différentes compétences vont être prises en charge par des **processus de régulation** (Cf. Figure 70). Selon le type de compétences disponibles dans le domaine source (analogiques, similaires ou spécifiques) et d'après la nature des compétences recherchées dans le domaine cible, quatre processus de régulation sont susceptibles d'être appliqués:

1. *Processus d'éradication de compétences spécifiques* : il s'agit d'écarter les connaissances qui ne sont plus d'aucune utilité dans la nouvelle situation de travail.
2. *Processus de création de compétences* : c'est le fait de développer de nouvelles compétences lorsque l'informaticien en manque pour traiter un problème original. C'est l'apprentissage de nouvelles connaissances.
3. *Processus d'ajustement et de maturation de connaissances similaires* : ce mécanisme doit permettre d'enrichir et/ou d'adapter les compétences aux exigences spécifiques du nouveau contexte de conception. Ce processus

fonctionne par "*généralisation d'expérience acquise*" et repose sur la théorie des schémas. On est alors dans le cas du réapprentissage.

4. *Processus de transposition de compétences analogiques* : cela revient à déplacer les compétences d'un contexte à un autre, sans modification, ni ajustement. Ce mode de fonctionnement est caractéristique du transfert analogique.

Ce schéma résume ces différents modes de régulation durant le transfert d'apprentissage.

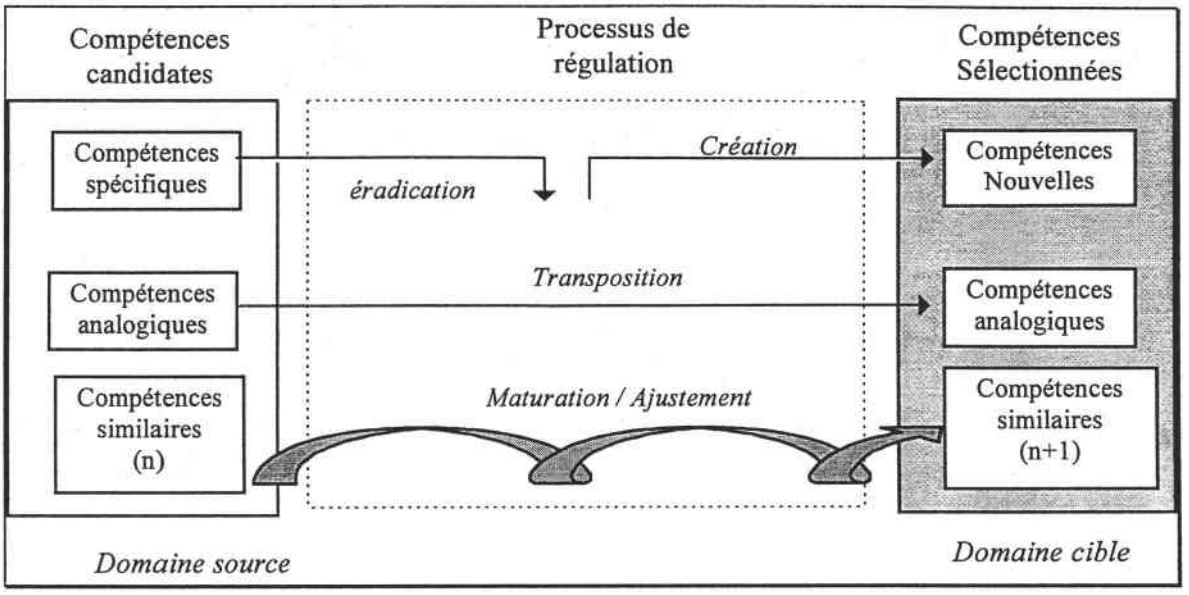


Figure 70
Les différentes modes de régulation des compétences durant le transfert d'apprentissage

Mais pour que le processus approprié de régulation se déclenche, il faut au préalable que l'informaticien se livre à un **diagnostic** de la situation cible. Durant cette phase, il extrait un ensemble d'informations et d'indices qui lui permet d'établir l'état de l'environnement.

Exemple d'indices recherchés :

- *Quels sont les objectifs de mon action ? : naviguer dans l'application ou concevoir une fenêtre ?*
- *Quel est sont les contraintes à respecter ?*

UNIVERSITÉ DE METZ BU Lettres S.C.D.

On parlera d'ailleurs plutôt de "*représentations fonctionnelles*" que "*d'état de l'environnement*" car élaborées pour une situation donnée et pour un temps limité. Cette représentation est alors comparée aux compétences stockées en mémoire. Cette confrontation a pour but de distinguer parmi les compétences candidates, celles qui sont susceptibles de convenir soit *parfaitement*, soit *partiellement*, soit *de manière erronée* à la situation cible.

Ainsi,

- si elles correspondent *parfaitement* au domaine cible ; ce sont alors des compétences analogiques qui seront transférées en l'état par raisonnement analogique ;
- si elles ne correspondent que *partiellement* ; un processus d'ajustement devra alors être appliqué sur les compétences similaires, pour qu'elles puissent être réutilisées convenablement dans le domaine cible ;
- si aucune ne convient, c'est le processus d'éradication/création de compétences qui sera déclenché.

Enfin, dernier mécanisme à intervenir dans ce transfert d'apprentissage, celui qui porte sur le **contrôle** des résultats obtenus. Ce processus évalue les résultats des actions par rapport aux objectifs attendus, et déclenche des corrections par rétroaction sur les différents mécanismes intervenant durant le transfert d'apprentissage. Ces corrections peuvent ainsi conduire l'informaticien à porter son choix sur une nouvelle compétence à acquérir, à appliquer une nouvelle procédure de régulation, ou bien, à réaliser un nouveau diagnostic de l'état du système.

Pour résumer, le transfert d'apprentissage implique différents processus dont le schéma ci-dessous rend compte :

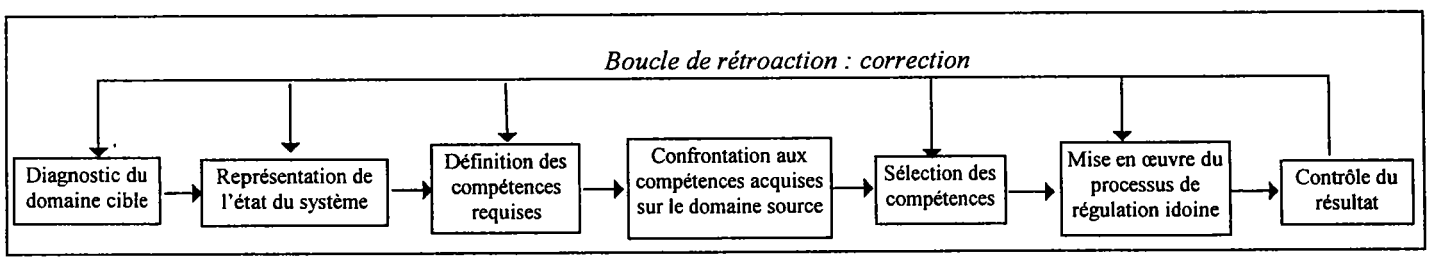


Figure 71
Processus cognitifs intervenant dans le transfert d'apprentissage

Dès lors, compte tenu de ces différents éléments, on peut supposer que les problèmes rencontrés par les novice résulteront :

- d'un diagnostic erroné de la situation de conception cible, causé vraisemblablement par le manque de pertinence des indices utilisés ;
- de l'approximation de la phase de définition des compétences requises ;
- du manque de rigueur et d'acuité du processus de sélection des compétences disponibles ;
- de l'application inadéquate des processus de régulation sur les compétences sélectionnées (choix d'un processus de transposition au lieu d'un processus de maturation, par exemple) ;
- d'une carence de compétences candidates suffisamment riches et variées, permettant de faire un choix optimal. A défaut de connaissances idéales, l'informaticien retiendra des compétences approximatives qui, malgré les ajustements qu'elles subiront, demeureront inappropriées au contexte cible ;
- enfin, du manque de fiabilité du processus de contrôle qui ne permet pas de repérer les erreurs générées par des transferts d'apprentissage négatifs, et de les rattraper par une boucle rétroactive.

En définitive, il semble donc que les erreurs commises par les informaticiens relèvent non seulement de l'incompatibilité des connaissances mobilisées avec les exigences du nouvel environnement technique, mais proviennent aussi du dysfonctionnement de certains mécanismes assurant le transfert d'apprentissage. Assimiler de nouvelles connaissances ne suffit donc pas pour travailler dans un contexte original, il faut aussi accommoder les procédures mentales qui agissent sur les conditions du réapprentissage et d'apprentissage.

Outre ces dysfonctionnements cognitifs, l'environnement technique affecte aussi la mise en œuvre de ces transferts de compétences. Nous allons en discuter dans la dernière partie.

1.3.2 Du rôle formateur de l'environnement sur les conduites cognitives de l'informaticien

Nous considérons que les environnements de conception sont des cadres de référence pour la construction et la mise en œuvre des raisonnements. Le système technique ne serait donc pas seulement une entité intermédiaire entre le sujet agissant et l'objet sur lequel porte son action. Il est aussi un artefact qui structure à la fois les formes de son travail et le contenu de ses savoirs.

Cette approche constructiviste et située de la cognition humaine suppose que les raisonnements s'élaborent dans le cadre des interactions homme-machine. Cela signifie aussi que les maquettes ne se développent pas de façon autonome au fur et à mesure que les étapes s'enchaînent, ni même que les environnements de développement emmagasinent passivement des données injectées par des sources externes (l'informaticien). Il s'agit d'une interaction continue entre d'une part, les structures cognitives de l'informaticien et ses actions sur l'environnement et, d'autre part, les renseignements (succès, échecs, difficultés) qu'il reçoit en retour. Chaque action sur l'environnement provoque ainsi, par réaction adaptative, une modification de la cognition qui, à son tour, sera responsable de la prochaine action sur le milieu. Donc, entre l'informaticien et l'environnement de développement, le rôle fondamental revient à un troisième facteur : les connaissances tirées de l'action. Dans ce cadre, on peut dire que la manipulation d'un dispositif a une "*visée pragmatique*" puisque les résultats qu'il donne sont utilisés pour engendrer des objectifs d'action, définir des plans d'action et générer de nouvelles solutions.

Certains environnements vont toutefois se révéler plus efficaces que d'autres dans cette forme d'assistance.

C'est notamment le cas de NSDK qui, par sa logique et ses modalités d'interaction, (*i*) permet à l'informaticien d'accéder directement aux résultats de ses actions, (*ii*) d'expérimenter des solutions nouvelles, (*iii*) d'organiser librement et opportunément son activité et, (*iv*) le cas échéant, de corriger sa conduite de travail.

En somme, les développeurs disposent par les "feed-back" du système –*simulation et présentation en temps réel de l'interface élaborée*– d'un compte rendu permanent sur la portée réelle de leurs interventions et sur la validité des solutions imaginées. L'informaticien prend directement conscience des erreurs commises et peut dès lors se

livrer à des régulations dynamiques sur les processus incriminés (ajustement de la solution ou choix d'une autre compétence).

Outre ces boucles rétroactives, un autre élément technique intervient dans la maturation des compétences professionnelles : c'est la reconnaissance que le système accorde aux actions de conception de l'informaticien. Dans la construction des maquettes avec **NSDK**, une part importante de cette activité est laissée à l'initiative du développeur. La conception graphique n'est pas un transfert unilatéral d'informations depuis un informaticien vers une machine, mais elle constitue un modelage mutuel au moyen d'actions conjuguées où les interventions du développeur donnent lieu à la réalisation effective du programme. Dans ce cadre, l'environnement de développement fonctionne comme un système de potentialités permettant de construire un mode de consonance au problème posé. Il ne s'agit donc pas d'appliquer les solutions déjà toutes faites au problème, mais bien de construire dynamiquement des solutions.

A l'inverse, **l'approche de type procédural de Pacbase** impose un cadre dont il est nécessaire de respecter la structure pour aboutir à l'architecture finale. En quelque sorte, l'environnement n'a pas besoin de l'implication cognitive de l'individu pour établir la procédure de résolution : elle préexiste aux interventions de l'informaticien. En revanche, ce dernier doit se soumettre aux règles techniques pour construire l'application et interagir avec le système. Il est pour ainsi dire l'auxiliaire passif du système. Dans le même ordre d'idée, l'environnement site-central ne tient pas compte de la singularité et des compétences particulières de l'informaticien : un même moule est ainsi imposé à tous les informaticiens, quels que soient leur niveau d'expérience et leur type de besoin. Mais il arrive un moment où les règles cessent d'être des repères encadrant l'activité, pour devenir des absolus. Ces ossatures de conception conduisent l'informaticien à adopter un mode de raisonnement rigide et sclérosé qui rendra d'autant plus difficile l'actualisation de leurs modes de pensée au nouvel environnement client serveur. En particulier, il lui sera difficile d'imaginer d'autres manières de concevoir et de résoudre les problèmes, d'autres façons de représenter et d'animer les interfaces, d'autres modalités pour interagir et agir avec l'environnement. En somme, toutes les caractéristiques qui ont été identifiées dans les modèles d'analyse.

Le paragraphe suivant propose une série de recommandations pour favoriser l'intégration des informaticiens dans le nouvel environnement.

1.4 RECOMMANDATIONS ERGONOMIQUES

Après avoir rappelé les principales difficultés observées chez l'informaticien novice, nous exposerons une série de recommandations ergonomiques dans le but de favoriser l'apprentissage et l'utilisation de l'environnement client serveur. Notons que ces propositions resteront volontairement très générales, compte tenu de la finalité de notre recherche et du matériel dont nous disposons. Cela dit, elles fournissent des axes de réflexion intéressants que des analyses ergonomiques plus spécifiques permettraient d'opérationnaliser.

1. Rappels des principales difficultés rencontrées par les novices lors de la conception informatique

Ce sont essentiellement des difficultés :

- à imaginer des représentations graphiques appropriées et originales ;
- à sélectionner des objets graphiques représentatifs et à les nommer par des libellés explicites ;
- à choisir les événements à associer aux objets ;
- à gérer et à anticiper les effets contextuels des événements programmés ;
- à organiser le contenu des fenêtres de manière équilibrée et fonctionnelle ;
- à structurer le dialogue homme-machine selon une logique asynchrone et événementielle ;
- à spécifier une démarche de maquettage personnelle en dehors de tout guidage formel de l'outil ;
- à positionner les objets graphiques sur les fenêtres en respectant les règles de conception (charte graphique) ;
- à disposer de modèles graphiques auxquels l'informaticien peut se référer durant la conception des fenêtres ;
- à dialoguer et à négocier avec de nouveaux partenaires de conception ;

2. Recommandations ergonomiques

Ces difficultés conduisent à l'augmentation de la charge mentale de l'informaticien. Ces problèmes peuvent également entraver la compatibilité H/M lorsque l'informaticien rejette sur le système, la responsabilité de ses propres défaillances.

Aussi, les recommandations proposées ont pour but d'atténuer ces difficultés et de favoriser des transferts d'apprentissage positifs entre les deux paradigmes de maquettage. Ces préconisations portent sur l'environnement de développement, la formation, la méthodologie de conception et les aspects sociaux du développement.

Environnement de développement

- *Promouvoir les techniques de simulation dans les formations - Redéfinir les messages d'erreur (plus explicites et illustrés d'exemples) - Enrichir la documentation en ligne par des cas pratiques.*

  Par ces biais, nous souhaitons développer l'apprentissage par l'action. Le développeur se base ainsi sur ces exemples pratiques et sur le résultat de ses simulations pour développer son interface. De même, les messages d'erreur clairement libellés, donnant les causes des erreurs et proposant des solutions exemplifiées, favorisentun apprentissage actif de la conception événementielle. Techniquement, ce système d'aide correspondrait au correcteur grammatical de "Word" qui justifie chaque faute identifiée par un exemple et une explication.

- *Fournir un assistant à la programmation de la maquette, en proposant des exemples pratiques.*

  Ces exemples se présenteraient sous la forme de gabarits dans lesquels l'informaticien peut insérer et spécifier ses propres paramètres. Néanmoins, ils seraient suffisamment ouverts et flexibles pour que l'informaticien puisse les faire évoluer dans le sens des nouvelles orientations de maquettage.

- *Fournir un assistant à la conduite de l'activité de maquettage.*

  Ce conseillé au maquettage rappellerait les étapes clefs à réaliser. Il jouerait ainsi le rôle de trame de conception dont l'absence sur NSDK a si fortement désorienté l'informaticien débutant. Concrètement, ce système déclinerait les différentes tâches à accomplir durant le maquettage, en proposant des exemples concrets d'application. Cet assistant soutiendrait l'activité de l'informaticien dès que celui-ci manifeste des difficultés pour préparer, organiser et conduire le déroulement de sa tâche. Ce "coach" pourra ainsi aider :

1. à construire des scénarii de maquettage ;
2. à habiller les fenêtres d'objets graphiques ;
3. à décliner la cinématique des fenêtres ;
4. à organiser les fenêtres selon les normes graphiques ;
5. à programmer les objets graphiques ;
    * par le choix des événements appropriés,
    * par la programmation des fonctions,
    * etc

Tout en étant suggestif, cet assistant ne doit pas être trop prescriptif. Sinon, l'informaticien aura l'impression d'être à nouveau confronté à un autre canevas de conception. On pourrait même imaginer un conseillé "intelligent" qui proposerait son aide dès que l'informaticien a dépassé un certain nombre d'erreurs de conception admises. Mieux, l'explication ou l'exemple, le style et le niveau d'aide seraient présentés en fonction de l'expérience de l'informaticien.

• *Franciser les termes anglais de l'environnement*

Afin de faciliter la sélection des événements de maquettage, il est nécessaire de trouver les traductions appropriées à tous les termes anglais de l'outil NSDK. Les interprétations approximatives ou erronées que peuvent en faire certains informaticiens conduisent à des choix inappropriés d'événements et donc, à des erreurs de programmation.

• *Fournir un assistant à la construction des fenêtres*

Afin d'épauler les informaticiens dans la construction de la maquette, un assistant proposerait plusieurs modèles de fenêtres réutilisables, dans lesquels les paramètres de présentation (distances standards entre les boutons, taille des fenêtres...) seraient déterminés selon les normes de la charte graphique du service.

• *Assister la recherche et la réutilisation des objets graphiques, des événements et des fonctions de programmation.*

L'informaticien est en effet souvent confronté à un ensemble d'entités sélectionnables (une centaine en tout) sur lesquelles il réalise des choix erronés faute d'un assistant performant. Cette aide devrait être capable de proposer des

événements ou des objets graphiques candidats en fonction d'un certain nombre de critères de maquettage indiqués par l'informaticien. Par exemple, selon le comportement qu'il souhaite faire adopter aux composants de la maquette (prééminence d'une fenêtre sur une autre), le système lui proposerait les événements "*Get-Locus*" et "*Lose-focus*".

Par ailleurs, pour éviter à l'informaticien de déporter manuellement le code d'un événement vers le programme général ; une procédure automatisée pourrait s'en charger.

- *Assister la gestion prévisionnelle et contextuelle des événements de l'interface.*

Il faut également aider les informaticiens à évaluer les effets contextuels des événements sur le reste de l'interface. Pour ce faire, nous préconisons d'implanter un système graphique où le changement d'état des objets serait visualisé sous la forme d'une arborescence logique : à chaque contrôle positionné sur la fenêtre est associé une flèche où l'on trouve à une extrémité l'objet déclencheur de l'action ; en vecteur, l'action déclenchée ; et à l'autre extrémité, le ou les objets touchés. Nous avons repris cette idée des brouillons des développeurs, qui utilisaient les flèches pour déterminer la nature des interactions entre les différents objets.

Formation

- *Réduire les délais entre formation et expérience* (actuellement, ceux-ci peuvent aller de 3 à 6 mois). Dans de telles conditions, il est très difficile pour l'informaticien de mettre en pratique les compétences qu'il a reçues.

- *Redéfinir les axes de formation en rapport avec les difficultés les plus souvent rencontrées par les développeurs*

Nous avons mis en évidence la réutilisation inappropriée de certaines stratégies de résolution procédurales dans l'environnement client serveur ; le débutant croyait reconnaître des similitudes entre des situations source et cible. La persistance de ces pratiques anachroniques entrave donc la conception d'interfaces réellement interactives : que cela soit au niveau de la structuration asynchrone du dialogue homme-machine, de la répartition des commandes sur les fenêtre (boutons ou menu), ou bien encore de la dénomination des objets graphiques.

Pour éviter de telles confusions, nous proposons d'axer les séances de formation sur des situations de conception site-central et client serveur relativement proches, et surtout, de caractériser les méthodes de résolution spécifiques à chaque contexte. De cette façon, l'informaticien pourra prendre conscience des particularités de chaque stratégie, et éviter ainsi de réutiliser, tout au moins volontairement, des procédures inadaptées.

Il serait aussi opportun de fragmenter la formation en quatre modules "*d'acquisition et d'apprentissage*" dont le contenu pédagogique reprendraient les grandes lignes de nos différents modèles d'analyse. Il est en effet fondamental que la formation permette au programmeur de se construire, non seulement une bonne représentation du fonctionnement du système (modèle d'interaction), mais qu'il soit aussi en mesure de décliner sa propre méthode d'intervention (modèle d'action) et de développer des stratégies de résolution pour résoudre des problèmes originaux (modèle de résolution). Il est utile, enfin, qu'il possède un grand nombre de modèles graphiques pour composer les fenêtres (modèle de référence).

- *Déceler les analogies appropriées entre les deux environnements et favoriser les transferts positifs.*

  On a découvert qu'une procédure de maquettage particulière à Pacbase (maquettage dynamique) était souvent citée comme référence à la construction des fenêtres sur NSDK. Il existerait donc une analogie entre ce mode de maquettage dynamique – *l'informaticien appelle directement les entités sur un écran vierge*– et le positionnement des objets sur les fenêtres par manipulation directe.

- *Enrichir et/ou consolider les modèles de référence graphiques.*

  Nous préconisons de sensibiliser le plus tôt possible les informaticiens à une culture de conception graphique (cela concerne surtout ceux qui travaillent encore dans l'environnement site-central) :

  - par des formations faisant un tour d'horizon des différentes possibilités de représentation graphique d'une maquette ;
  - par la distribution de documents dans lesquels seraient exposés différents modèles d'interfaces graphiques ;

- par l'intégration de l'environnement site-central sur les stations de travail client serveur. Les systèmes de développement (Pacbase, DB2...) seraient alors accessibles *via* des fenêtres émulées à l'écran. Les logiciels bureautiques de cette station de travail (Word, Powerpoint...) pourraient être employés pour réaliser la documentation de l'application. De cette manière, l'informaticien aurait la possibilité de se familiariser avec les métaphores graphiques et les modalités d'interaction particulières de cet environnement (manipulation de la souris, navigation dans le système, compréhension de la logique événementielle).

Méthode de maquettage

- *Définir un ordonnancement temporel des tâches clefs à effectuer durant le maquettage et proposer des procédures de résolution susceptibles d'y être employées.*

  Cet ordonnancement s'inspire de celui déjà proposé dans l'assistant au maquettage. L'idée est de présenter des prémisses de conduite que l'informaticien pourra s'approprier et enrichir au gré de ses expériences.

Documentation des outils de développement

- *Proposer deux types de documentation, se basant respectivement sur la logique d'utilisation de l'informaticien et sur la logique de fonctionnement du système.*

  Richard (1983) a montré qu'une documentation comprend toujours deux types d'informations nécessaires à l'utilisation d'un système : i) des informations sur **la logique d'utilisation** du dispositif qui expliquent comment utiliser le système pour obtenir un résultat donné (par exemple, *"pour obtenir tel résultat, choisir telle commande"*), et ii) des informations sur **la logique de fonctionnement** du dispositif qui expliquent l'effet des commandes à travers les modifications qu'elles produisent dans l'état du système (par exemple, *"telle commande produit tel résultat"*).

  Le premier type d'information est utile pour une utilisation rapide du système. Le second type permet à l'utilisateur de construire une bonne représentation du fonctionnement du système.

Pour favoriser l'apprentissage rapide et efficient du système, nous recommandons de partager la documentation entre une logique d'utilisation, d'une part, et une logique de fonctionnement, d'autre part.

- Dans une première partie, seraient ainsi exposées les informations relatives à la logique d'utilisation. Par exemple, « *Pour sélectionner un événement, faire telle séquence d'actions avec telles commandes* », ou encore « P*our nommer un objet graphique, faire telles actions avec telles commandes* ».
- Dans la seconde partie, ces mêmes renseignement seraient présentés selon une logique fonctionnelle. Par exemple, « telle commande *(cliquer deux fois sur le bouton droit de la souris lorsqu'elle est positionnée sur un objet graphique)* permet de faire telle action (*ouvre sa boîte de dialogue afin de nommer et de paramétrer le bouton*) ».

Ces documentations devront veiller également à proposer un maximum d'exemples réutilisables et exploitables par l'informaticien dans sa tâche de conception.

Les relations humaines

- *Développer la communication intra-projet et interprojet*

Autre changement majeur intervenu avec l'évolution technique, l'augmentation des intervenants impliqués dans le projet. L'informaticien doit collaborer avec un plus grand nombre d'acteurs dont les compétences couvrent divers domaines techniques et fonctionnels de la conception. En outre, pour pallier la carence des librairies de NSDK, les informaticiens échangent davantage avec les autres équipes projet, afin de réutiliser leurs fonctions de développement.

Pour optimiser ces interactions, nous proposons de reporter les compétences de chaque partenaire dans un annuaire professionnel (classé par fonction et par nom). Par ailleurs, un serveur et une messagerie Intranet, rendant compte de tous les projets réalisés par les différentes équipes, établiraient les bases d'une bourse d'échange informatique. En somme, les informaticiens proposeraient, rechercheraient et "troqueraient" des entités informatiques.

2. CONCLUSION GÉNÉRALE

Les principaux acquis de cette recherche résident d'une part, dans la mise en évidence des modèles mentaux impliqués dans le maquettage informatique et, d'autre part, dans l'analyse des facteurs responsables des difficultés rencontrées par les informaticiens débutants désirant s'approprier le nouveau dispositif de conception.

a) Les modèles mentaux impliqués dans le maquettage informatique

Il ressort ainsi que :

1. L'organisation de l'activité de conception varie en fonction du contexte de sa production. En particulier, le site-central impose une organisation hiérarchisée de l'activité alors que l'environnement client serveur permet une gestion opportuniste et située du développement.

2. Si les dispositifs techniques induisent le choix de certaines stratégies de conception, il n'en demeure pas moins que l'expérience préalable de l'informaticien conditionne tout autant fortement leur mise en œuvre.
   En particulier, les caractéristiques techniques et fonctionnelles du nouveau dispositif exigent un remaniement des pratiques cognitives de résolution, comme :
   * passer de la représentation d'un problème de transformation d'états sur Pacbase, à la représentation d'un problème de conception avec NSDK ;
   * passer d'une résolution analogique dans le contexte de conception répétitif de Pacbase, à une résolution par construction et expérimentation de solutions dans le contexte innovant de NSDK ;
   * passer des traitements dirigés par les concepts sur Pacbase vers les traitements dirigés par les donnée avec NSDK :

   Néanmoins, les acquis cognitifs de l'informaticien peuvent également orienter les modalités de la résolution :
   * ainsi, le processus d'évaluation des solutions "*en largeur d'abord*" ne peut se dérouler que si l'informaticien dispose d'expériences riches et diversifiées ; alors que l'évaluation "*en profondeur d'abord*" est plus représentative des novices qui ne peuvent compter que sur très peu de solutions.

3. Les relations entre l'homme et la machine évoluent. Les analyses effectuées abondent dans ce sens puisqu'elles ont montré que les distances sémantiques, opératoires et articulatoires étaient plus importantes dans le cadre de l'interaction homme-machine du site-central que dans celui du client serveur. Ces écarts créent les conditions d'une incompatibilité cognitive et d'une surcharge mentale dont les effets se mesurent à travers le manque d'implication de l'informaticien dans la conception.

4. La nature des modèles auxquels se réfèrent les informaticiens pour construire leur maquette se transforment. Pour être capable d'imaginer des interfaces graphiques originales, l'informaticien n'a pas d'autre moyen que de délaisser ses anciens modèles de conception, étroitement associés à Pacbase, pour se consacrer à la construction de nouvelles représentations fonctionnelles, ouvertes sur le monde extérieur.

Par ailleurs, nous avons sommairement fait état de transformations socio-organisationnelles générées par l'événement du système client serveur : élargissement des domaines d'activité couverts par l'informaticien (analyse + programmation), augmentation du nombre d'intervenant dans la conception, implication plus forte de l'utilisateur final... En somme, tout un ensemble de modifications qui avait conduit au repositionnement de l'informaticien dans le processus de conception.

Finalement, ces différentes analyses suggèrent que les informaticiens qui sont confrontés à l'évolution technologique de leur système informatique font face à l'articulation de trois réalités : cognitives, organisationnelles et socio-professionnelles.

- d'une part, les changements techniques obligent les individus à apprendre d'autres raisonnements de conception et d'autres manières d'interagir avec les outils de développement informatiques ;
- d'autre part, les situations de travail évoluent sur le plan du contenu de la tâche et de leur organisation ; ce qui amène les informaticiens à revoir leur manière de structurer leur activité ;
- enfin, l'évolution technique conduit les informaticiens à développer d'autres comportements sociaux et relationnels, fondés sur davantage de coopération et de négociation.

Aussi, le changement technique ouvre un nouvel espace de représentations (cognitives, organisationnelles et socio-professionnelles) qui implique un repositionnement des informaticiens en termes de raisonnement, d'identité et de culture. La conception informatique est ainsi soumise à une dynamique de changement, faisant coïncider mutations techniques, acquisitions mentales et transformations sociales. Dans ces conditions, l'informaticien doit faire évoluer ses compétences dans le sens des exigences particulières de la nouvelle situation de développement. Seulement, on a vu que cette opération était loin d'être un processus aisé pour le débutant, qui commettait de nombreux impairs dans le nouvel environnement. Deux séries d'explications ont été avancées : d'une part, la réutilisation de compétences inadaptées, et d'autre part, l'incidence de l'environnement technique source sur les conduites de l'informaticien, qui rend difficile son adaptation au nouveau système.

b) Les facteurs responsables des difficultés des novices

Concernant les transferts d'apprentissage, un modèle explicatif des dysfonctionnements de la réutilisation a été proposé. Divers processus étaient incriminés :

- le diagnostic erroné de la situation cible ,
- le manque de rigueur et d'acuité du processus de sélection des compétences sources ;
- l'inadéquation des processus de régulation appliqués aux compétences sélectionnées ;
- le déficit de compétences suffisamment riches et variées permettant de faire un choix optimal ;
- le manque de fiabilité du processus de contrôle.

En définitive, les difficultés rencontrées par les informaticiens relèvent non seulement de l'incompatibilité des connaissances mobilisées avec les exigences du nouvel environnement, mais proviennent aussi de l'inadéquation et/ou d'un dysfonctionnement de certains processus intervenant dans le transfert d'apprentissage. Assimiler de nouvelles connaissances ne suffit donc pas pour travailler dans un

contexte de travail original, il faut aussi accommoder les mécanismes qui agissent sur les conditions du réapprentissage et d'apprentissage.

Pour ce qui est de l'impact du dispositif technique sur les conduites de conception, on a montré que si l'environnement site-central enfermait l'informaticien dans une sorte de dépendance technique, réglant, ordonnant et évaluant l'ensemble de ses actes ; le système client serveur, en revanche, présentait une attitude plutôt "libérale" en laissant à l'individu l'initiative d'inventer ses propres solutions et d'agencer librement son activité. Mais, justement, la principale difficulté pour le novice résidait dans sa capacité à s'émanciper techniquement et fonctionnellement, c'est-à-dire à s'affranchir de la coupe de l'ancien système pour exister "cognitivement" dans le nouvel environnement.
En conséquence, l'environnement technique peut être considéré comme le moule fondateur des conduites de l'informaticien. Il change l'état des relations entre le développeur et son objet de travail. Il instaure des "règles de jeu" nouvelles avec lesquelles l'informaticien doit composer pour trouver un accord sur un état de mise en forme du monde : la maquette finale. Il établit aussi des exigences originales auxquelles l'informaticien doit répondre par l'acquisition de nouvelles compétences et/ou par l'adaptation de raisonnements plus anciens.

Enfin, le système technique redessine l'espace socio-organisationnel de la conception en redéfinissant la nature des interactions et le rôle de chaque acteur au sein du processus de développement : informaticiens, mais aussi utilisateurs, partenaires de conception et décideurs. Dans cette perspective, le changement d'outil provoque aussi la reconfiguration de la dimension collective de la conception. On passe d'une approche centralisée du développement de type site-central (un informaticien → un programme) à une conception socialement distribuée sur le client serveur (plusieurs intervenants → un programme).

En conséquence, ce qui détermine l'appropriation des nouvelles technologies informatiques par les informaticiens débutants, c'est moins leur complexité technique intrinsèque que la série de ruptures qu'elles engendrent *i)* sur les modalités de

raisonnement, *ii)* sur les conditions de contrôle de l'activité et *iii)* sur les principes de coopération au sein du cycle de conception.

c) Apports et perspectives de recherche

Dans notre recherche, les processus cognitifs impliqués dans le maquettage sont abordés dans leur contexte réel de production, à l'inverse des recherches qui les étudient en situation artificielle. De même, l'analyse de la tâche de maquettage est originale puisqu'il n'existe pas, à notre connaissance, d'études sur ce problème. Toutefois, nous considérons que les clivages fréquemment dénoncés entre les recherches expérimentales et empiriques correspondent davantage à des points de vue différents sur les méthodes à employer qu'à des oppositions sur les modèles de fonctionnement identifiés. Ainsi, ces approches, loin de souligner les divergences entre leurs conclusions, indiquent leurs complétudes. Seules, elles ne peuvent pas aborder les compétences de la conception informatique dans leur globalité et leur complexité. Par contre, mises en perspective dans une approche pluridisciplinaire, l'analyse expérimentale et l'approche empirique peuvent identifier et comprendre les problèmes relatifs à cette activité informatique. L'une nourrissant l'autre de ces découvertes, et l'autre validant ces données.

Quand aux perspectives sur lesquelles peuvent déboucher cette recherche, il semble que trois types de travaux soient envisageables :

Le premier, orienter sur la recherche expérimentale, viserait à valider le modèle explicatif des transferts d'apprentissage par un protocole expérimental.
Il s'agirait de former des personnes, non informaticiennes, à l'utilisation de deux logiciels informatiques. Dans une première séance de formation à un logiciel A, on focaliserait l'apprentissage de trois échantillons sur différents types de compétences : compétences spécifiques à ce logiciel, compétences identifiées comme analogues à deux environnements (A et B), et compétences susceptibles d'être réutilisées dans le logiciel B, mais avec des réajustements préalables.
Quelques temps après, on rappellerait ces personnes et on leur ferait subir une seconde séance de formation à un logiciel B. Cette fois-ci, le contenu de la formation serait identique aux trois échantillons. Une tâche informatique leur serait proposée, mais sans l'aide de ce logiciel. En fait, on leur demanderait d'imaginer ce qu'ils feraient

s'ils avaient l'outil à leur disposition (test du magicien d'Oz) : quelles seraient les commandes qu'ils utiliseraient, les séquences d'action qu'ils planifieraient, les stratégies qu'ils emploieraient, les informations qu'ils rechercheraient, les parties du logiciel qu'ils emploieraient... Par cette méthode, on suppose que l'informaticien ferait appel à toutes les compétences qu'il a sa disposition ; c'est-à-dire non seulement celles du logiciel B, mais aussi celles acquises en A.
L'analyse des données permettrait d'identifier expérimentalement l'origine des compétences réutilisées, leur nombre, leurs éventuelles transformations, les bonnes ou mauvaises associations, les sources d'erreurs... On demanderait également à l'individu de réagir à ces données et d'expliquer pourquoi il a été amené à utiliser tel type de connaissances plus que tel autre type plus adapté. Bien évidemment, le protocole d'une telle expérience serait à affiner afin d'obtenir des analyses plus tangibles.

La deuxième perspective, centrée sur la recherche appliquée, consisterait cette fois à la prise en compte des recommandations ergonomiques pour l'aménagement de l'environnement d'accueil client serveur. Les conseils que nous avons prodigués (voir partie discussion générale) touchent aux aspects logiques, graphiques et fonctionnels de NSDK, à la formation, aux méthodes d'organisation de l'activité et à la documentation. Sans entrer à nouveau dans les détails de ces recommandations, il faut néanmoins souligner le rôle fondamental d'assistant que devra jouer le dispositif pour former et guider l'informaticien dans l'apprentissage de la conception. En aucun cas le système ne doit diriger les conduites de l'individu. Bien au contraire, il assiste l'informaticien dans sa tâche et reste suffisamment flexible et ouvert pour accueillir des profils variés de développeur. Visant aussi la compatibilité homme-machine-tâche aux niveaux cognitifs, perceptifs, opératoires et linguistiques, le dispositif sera dès lors tout à fait adapté aux informaticiens et aux caractéristiques de leur activité.

La troisième et dernière approche concerne l'ouverture du champ d'étude de la conception informatique à des disciplines connexes des sciences humaines, comme la sociologie du travail. En effet, comme il en a été fait état plus haut, tout changement technique passe non seulement par la découverte et l'acquisition de nouvelles capacités collectives, mais implique aussi de nouvelles façons de raisonner ensemble ;

c'est-à-dire de partager et d'échanger des connaissances et des informations à travers un nouvel espace de référence. Changer de techniques, c'est donc aussi apprendre à coopérer autrement, à établir de nouveaux modèles de relation et de négociation. Il ne s'agit plus seulement de s'approprier individuellement une nouvelle technologie, mais de s'engager, par son intermédiaire, dans un processus de conception collective où les acteurs construisent et partagent des représentations et un univers commun. Dès lors pour appréhender ces phénomènes socio-collectifs,les contributions de la sociologie du travail nous semblent très fertiles. On rappellera d'ailleurs que la "cognition située" est un concept qui a été proposé par les sociologues du travail[62]. Mais derrière cette vision œcuménique, nous voulons défendre l'idée d'une approche réellement pluridisciplinaire de la conception informatique, tant les domaines d'activité qu'elle touche et les dimensions qu'elle recouvre sont nombreux, complexes et variés. Pour terminer, nous dresserons rapidement les contours de cette discipline axée principalement sur l'analyse des effets du changement technique :

- Concernant ses objectifs, elle viserait :
  - à rendre compte de l'activité d'individus réunis au sein d'équipes de travail, impliqués dans une activité complexe de résolution de problème informatique, et confrontés à un environnement technique mouvant,
  - à s'interroger sur les capacités d'adaptation et d'évolution de personnels engagés dans un processus de changement, et plus spécifiquement, d'accession à de nouvelles technologies de travail,
  - à analyser les facteurs susceptibles d'entraver, ou au contraire de favoriser ce changement,
  - à rendre compte des aspects collectifs de la conception et de ses mutations dans le cadre des évolutions technologiques.

- Concernant ses méthodes, elle axerait principalement son intervention sur l'observation par une approche située de l'activité de travail. Néanmoins, elle ne rejetterait pas les apports des recherches expérimentales.

[62] On pourra se référer au périodique : Sociologie du Travail (1994) "*Travail et Cognition*", Numéro spécial, n° 4/94.

- Concernant ses enjeux, cette compréhension globale du travail devrait permettre une meilleure coopération entre tous les acteurs du changement (décideurs, utilisateurs finaux, informaticiens...) afin de mieux concevoir la place de l'homme dans l'univers technique et social ; c'est-à-dire de contribuer à « *l'humanisation du travail (...), tâche permanente destinée à accompagner un développement technologique continu* [car] *les problèmes relatifs au travail apparaissent et croissent beaucoup plus vite que les techniques développées pour résoudre ces problèmes, ou les solutions techniques adoptées pour leur apporter une compensation* » (Lutz, 1991, cité par Teiger 1993, p 95).

Cependant, nous avons conscience que les contraintes économiques et organisationnelles inhérentes aux entreprises limitent ce genre d'intervention. Aussi, les résultats présentés dans cette thèse pourront sans doute être utilisés pour une réflexion plus globale sur les conditions de mise en œuvre d'un transfert technologique dans le cadre de la conception informatique.

BIBLIOGRAPHIE

- Adelson B., Soloway E. (1988) "The role of domain experience in software design". In *IEEE Transactions on software Engineering*, 1985, pp 1351-1360.
- Agro L, Cornet A. , Pichault F. (1995a) "L'implication des utilisateurs dans les projets informatiques : un scénario en quête d'acteurs". In *Annales des Mines "Gérer et Comprendre"*, numéro 41, Paris, Edition ESKA, pp 33-44.
- Agro L, Cornet A. , Pichault F. (1995b) "Modalités de l'implication des utilisateurs et représentations des chefs de projets informatiques". In *Actes du 10° colloque Européen en Informatique et Société (CREIS) "Responsabilités sociales et formation des acteurs de l'informatisation"*, 5-7 juillet 1995, Belgique, Namur, pp 131-139
- Alsène E. (1990) "Les impacts de la technologie sur l'organisation". In *Sociologie du travail*, n°3/90, pp 321-337
- Alter N. (1996) "*Sociologie de l'entreprise et de l'innovation*". Paris, Puf.
- Anderson J.R., Thomson R. (1989) "Use of analogy in a production system architecture". In Vosniodou S., Ortony A. (Eds.), *"Similarity and analogical reasoning"*. Cambridge University Press
- D'Astous P., Détienne F., Robillard P.N., Visser W., (1997) "A coding scheme to analyze activities in technical review meetings", In *The 10th Workshop of the Psychology of Programming Interest Group (PPIG98).* Milton Keynes, UK.
- Bainbridge L. (1990). "Verbal protocol analysis. Evaluation of human work: a pratical ergonomics methodology". In J.R Wilson J.R., E.N Corlett (Eds). London : Taylor and Francis, pp 161-179.
- Barcenilla J. (1993) "*Etude sur la compréhension et le suivi d'instructions lors de l'apprentissage de dispositifs techniques*". Thèse en Psychologie des processus cognitifs, Décembre 1993, Université de Paris VII.
- Barcenilla J., Tijus C. (1998) "Acquisition, description et évaluation des savoir-faire : un point de vue cognitif". In *Connexions "Les Compétences Professionnelles"*, n°70, pp 31-46.
- Bastien J.M, Scapin D. (1993) "*Ergonomic Criteria for the Evaluation of Human-Computer Interfaces*". Rapport Technique n° 156, INRIA Rocquencourt.
- Batra D., Antony R. (1994) "Effects of data model and task characteristics on designer performance : a laboratory study". In *International Journal-Human Computer Studies*, 41, pp 481-508.
- Bellamy R. (1992) "Re-structuring the programmer's task" in *International Journal Man-Machines Studies*, 37, pp 503-527
- Bellamy R. (1994), " What does pseudo-code do ? A psychological analysis of the use of pseudo code by experienced programmers". In *Human Computer Interaction*, vol 9, pp 225-246.

- Berlioux J-M (1995) "Concevoir les applications de gestion à interface graphique". In *Génie Logiciel* , n°16, pp 3-10.
- Birrien J-Y. (1992) "*Histoire de l'informatique*" ; Que sais-je ?, Paris, PUF.
- Bisseret A. (1985) "Les hommes et l'informatique" , In *Le monde de l'informatique* , 28 octobre, pp 43-46.
- Bisseret (1987) "Les activités de conception et leur assistance". In *Bulletin de liaison de la recherche en Informatique et en Automatisme*, n° 115, INRIA Rocquencourt.
- Bisseret A., Figeac-Letang C., Falzon P. (1988) "Modélisation des raisonnements opportunistes : l'activité des spécialistes de régulation des carrefours à feux". In *Numéro spécial de la Psychologie Française "la Psychologie de l'expertise"*, Tome 33-3, pp 161-170.
- Bizec J-F. (1981) "*Les transferts technologiques*". Paris, Puf, Que sais-je.
- Blackwell A (1996) "*Metacognitive theories of visual programming : what do we think we are doing ?* ". In VL'96.
- Blandford A.E., Barnard P.J. (1995) "Using Interaction Framework to guide the design of interactive systems". In *International Journal Human-Computer Studies*, 43, pp 101-130.
- Bobillier-Chaumon M-E., (1996)."Aspects psycho-cognitifs de la conception orientée-objet". In *Actes du 2° colloque "Jeunes Chercheurs en Sciences Cognitives'* Giens, ARC, pp 272-275.
- Bobillier Chaumon M-E., Brangier E., (1996). "Analyse psycho-ergonomique de l'activité de maquettage dans un contexte de migration technologique". In *Actes du Colloques IHM'96*, Grenoble 15-17 septembre, pp 3-8.
- Bobillier Chaumon M-E., (1997). "D'une dépendance technique à une interdépendance collective : l'exemple d'un collectif de travail dans un contexte de migration technologique". In Brangier E., Dubois N., Tarquinio, C.,.(Eds), *Actes du Colloque "Compétences et Contextes Professionnels : perspectives psychosociales,"* Metz 19-20 juin 1997, (173-181).
- Bobillier Chaumon M-E, (1998) "Changements techniques et évolution des compétences". In *Connexions "Les Compétences Professionnelles"*, n°70, pp 165-180.
- Boreham N.C.; Patrick J. (1996) "Diagnosis and Decision-Making in work situations. An Introduction". In *Le Travail Humain*, Tome 59, n°1/1996, pp 1-4.
- Brangier E. (1991) "*La modélisation de la cognition dans l'élaboration d'un système expert*". Thèse de doctorat de Psychologie du travail, Université de Metz.
- Brangier E., Bobillier Chaumon, M-E., (1995).- "La programmation procédurale et orienté-objet : une étude d'ergonomie cognitive comparée" - In *Actes du Colloque JAVA'95 (Journées Acquisition, Validation, Apprentissage)*, Grenoble, CNRS - INRIA, 25.
- Brangier, E., Bobillier Chaumon, M-E., (1996).- "Réflexions sur l'intervention en ergonomie de la programmation" .- In R. Patesson (Ed.) *Actes du colloques de la SELF "Intervenir par l'ergonomie"* , Bruxelles. Vol 2, pp 176-183.
- Brangier E., Tarquinio C. (1997) "Le rôle du contexte professionnel dans la définition de la compétence et de son usage". In *Brangier E., Dubois. N., Tarquinio C.,.(Eds), Actes du*

Colloque "Compétences et Contextes Professionnels : perspectives psychosociales," Metz 19-20 juin 1997, pp 84-93.

- Breton P. (1987) "*Histoire de l'informatique*". Paris, La découverte.
- Bucciarelli LL (1988) "An ethnographic perspective on engineering design". In *Design Studies*, 9, 3, pp 185-190.
- Buckland M.K., Liu Z. (1995) "History of information science". In *Annual Review of Information Science and Technology* (Arist), vol 30, pp 385-416.
- Burkhardt J-M., Détienne F. (1995) "La réutilisation de solution en conception de programmes informatiques". In *Psychologie Française*, 40-1, pp 85-98.
- Burnett M.M., Baker M., Bohus C., Carlson P., Yang S. et van Zee P. (1995) "*Scaling up visual programming languages*". IEEE Computer.
- Canas J.S., Bajo M.T., Gonzalvo P. (1994) "Mental Models and Computer programming". In *International Journal Human-Computer Studies*, 40, pp 795-811.
- Caverni J.P, Bastien C. (1988) "*Psychologie Cognitive : modèles et méthodes*". Grenoble, PUG.
- Cavestro W. (1987) "Nouveaux langages, Nouveaux travail". In P. Bernoux, W. Cavestro, B. Lamotte, J.F Troussier (Eds) "*Technologies nouvelles, Nouveau travail*", Paris, coll. Recherche, Centre fédéral, pp 71-74.
- Chao B.P. (1987) "Prototyping a dialogue interface : a case study". In G. Salvendy (Ed) "*Cognitive Engineering in the design of Human Computer Interaction and Experts Systems.* Amsterdam", Elsevier Sciences Publications, pp 357-363.
- Chatel S., Détienne F. et Borne I. (1992) "Transfer among programming languages : an assesment of various indicators". In *5° Workshop of the Psychology of Programming Interest Group* (PPIG), 10-12 déc 1992.
- Chatel S. (1997) "*L'acquisition d'un langage de programmation orienté-objet : Smalltalk-80*" , Thèse de doctorat de Psychologie, Université Paris VIII.
- Chauvin C. (1991) "Maintenance de logiciels : les apports de la recherche". In *Performances Humaines et Techniques* : *La maintenance* , n° 55, Toulouse, Octares Edition, pp 13-17.
- Cohen J.C (1990) "*Les infomanageurs ou comment maitriser l'informatisation ?*". Paris, Les Editions d'Organisation.
- Cohendet, Lherena, Mutel, (1992) "Flexibilité et mise en cohérence des données de production". In G. De Terssac et P. Dubois (Eds) "*Les nouvelles rationnalisations de la production*". Toulouse, Cépadués-Éditions, p 25-41.
- Conein B., Jacopin E., (1994) "Action située et cognition. Le savoir en place". In *Sociologie du travail*, n°4/94, pp 475-500..
- Corniou J.P, Hattab N. (1996) "L'analyse de la valeur des applications informatiques". In *l'informatique professionnelle*, n°147, octobre 1996, pp 37-49.
- Coulanges B. (1993) "Impact économique de la réutilisation". In *Le génie logiciel et ses applications* (6th Int Conf). 15-19 nov 1993. Le CNIT Paris La Défense, pp 305-314.

- Darses F. (1990) "*Gestion des contraintes au cours de la résolution d'un problème de conception de réseaux informatiques*". Rapport de Recherche n° 1164, INRIA Rocquencourt.
- Darses F., Falzon P. (1996) "La conception collective : une approche de l'ergonomie cognitive". In G. de Terssac, E. Friedberg (Eds) "*Coopération et Conception*", Toulouse, Octares Edition, pp 123-147.
- Davies S.P. (1991) "Characterizing the program design activity : neither stricly top-down nor globaly opportunistic" . In *Behaviour and Information Technology"*, n° 10, pp 173-190.
- Davies S.P. Castell A.M. (1991) "From individuals to group through artifacts. The changing semantics of design in softaware development". In *NATO Workshop on "User-centred requirements for software engineering environments"*, Bonas (France), 5-10 sept. 91, pp 1-25.
- Delmond D.H. (1995) "Perception, rôle et mode de gestion de l'informatique dans les entreprise : le cas des services études". In *Actes du 10° colloque Européen en Informatique et Société (CREIS), "Responsabilités sociales et formation des acteurs de l'informatisation"*, 5-7 juillet 1995, Belgique, Namur, pp 57-68.
- Desaintquentin et B. Sauteur (1991) "*L'informatique éclatée : tendances actuelles 1991-1993"*. Paris, Masson.
- Détienne F. (1986), "*La compréhension de programmes informatiques par l'expert : un modèle en termes de schémas*" Doctorat de Psychologie. Université de Paris V.
- Détienne F. (1988) "Une application de la théorie des schémas à la compréhension des programmes". In *Le Travail Humain*, V. 51, n°4, pp 335-350.
- Détienne F. (1989a) "L'approche de l'ergonomie cognitive en programmation informatique". In *Actes du colloque Le génie logiciel et ses applications*, 4-8 décembre 1989, pp 167-179.
- Détienne F. (1989b) "Une revue des études psychologiques sur la compréhension des programmes informatiques". In *Techniques et Sciences en Informatiques*, volume 8, (1), pp 5-20.
- Détienne F., (1990a) "Un exemple d'évaluation ergonomique d'un système de programmation orienté-objet : le système O2". In *Actes du colloque Ergo'IA*, pp 64-74.
- Détienne F., (1990b) "Expert programming Knowledge : a schema-based approach". In J.M Hoc, T.RG Green, R. Samurcay, D.J Gilmore. (Eds) "*Psychology of programming*", London, Academic Press, pp 205-222.
- Détienne F. (1990c) "Program Understanding and Knowledge organization : the influence of acquired schemata". In *Cognitive Ergonomics*, London, Academic Press, pp 245-256.
- Détienne F. (1991a) "Constraints on design : language, environment, code representation". In *NATO Workshop on User-centred requirements for software engineering environments,* Bonas (France), 5-10 sept. 91, pp 315-322.
- Détienne F. (1991b) "Reasonning from a schema and from an analog in software code reuse". In Koenemann-Bellibeau, T. Moher, S.P Robertson (Eds) "*Empirical Studies of Programmers : 4th workshop"*, pp 5-22.

- Dezalay T. (1995) "L'approche techniciste des systèmes informatiques et ses limites". In *Actes du 10° colloque Européen en Informatique et Société (CREIS) "Responsabilités sociales et formation des acteurs de l'informatisation"*, 5-7 juillet 1995, Belgique, Namur.
- Dobrov G.M. (1979) "La technologie en tant qu'organisation". In *Revue internationale des Sciences Sociales,* Col. XXXI, n° 4.
- Dubost J (1987) "*L'intervention psychosociologique*". Paris, Puf.
- Ducateau C.F. (1995) "Contribution à la réflexion sur les responsabilités sociales de l'informaticien concepteur de logiciel". In *Actes du 10° colloque Européen en Informatique et Société (CREIS), "Responsabilités sociales et formation des acteurs de l'informatisation"*, 5-7 juillet 1995, Belgique, Namur, pp 149-157
- Durand C. (1992) "Les transferts internationaux de technologie". In *Sociologie du travail,* n°2/92, pp 139-152
- Eyraud F., Iribarne A., Maurice M., (1988) "Des entreprise face aux technologies flexibles : une analyse de la dynamique de changement". In *Sociologie du travail,* n°1-88, pp 55-77.
- Falzon P. (1995) "Les activités de conception : réflexions introductives" in *Performances Humaines et Techniques : L'activité des concepteurs,* n° 74, Toulouse, Octares Edition, pp 6-11
- Falzon P., Visser W. (1989) "Variations in expertise : implications for the design of assustance systems". In G. Salvendy, M. Smith (Eds)", "*Designing and using human-compouter interfaces and knowledge based systems*". Amsterdam. Elsevier.
- Fischer G. (1991) " Putting the owners of problems in charge with domain-oriented design environments". In *NATO Workshop on User-centred requirements for software engineering environments,* Bonas (France), 5-10 sept. 91, pp 297-307.
- Fischer G.N., Brangier, E., (1990).- Implantation d'un système expert : stratégie d'appropriation et changement organisationnel.- In *Actes du colloque "L'économique et l'intelligence artificielle", CECOIA2*, Paris, AFCET, pp 89-94.
- Fischer G.N. (1997) "*La psychologie sociale*", Paris, Points.
- Fortin R., Rousseau R. (1994) "*Psychologie cognitive : une approche de traitement de l'information*". Université du Québec, Canada, Télé-Université.
- Frechin C. (1994) "Le métier d'informaticien : un excellent poste d'observation de l'entreprise". In *Le monde informatique* du 15 avril 1994, p 31.
- Frese M. et Hesse W. (1995) "Analyse psychologique du travail des développeurs". In *Forum Logiciel*, Août 1995, pp 2-10.
- Friedberg E. (1988) "La participation" In *L'analyse sociologique des organisations*, Paris, L'Harmattant.
- Frohlich D., Luff P. (1989) "Conversational Ressources for situated action". In *CHI'89 Proceedings*. ACM, pp 253-258.
- Galand B. (1995) "La ville artificielle. Les réseaux : de la responsabilité des informaticiens à celle des utilisateurs". In *Actes du 10° colloque Européen en Informatique et Société (CREIS), Responsabilités sociales et formation des acteurs de l'informatisation*, 5-7 juillet 1995, Belgique, Namur, pp187-197

- Gammack J., Young R.M., (1984) "Psychological techniques for eliciting expert knowledge". In M.A Bramer (Ed) *"Research and development in expert systems"*. Proceedings of the fourth technical conference of the british computer society specialist on expert systems, University of Warwick, (18-20 déc 1984), pp 105-112.
- Gentner D. (1989) "The mechanisms of analogical learning". In S. Vosniadou, A. Ortony (Eds) *"Similarity and analogical reasoning"*. Cambridge, University Press, Chap. 7, pp 199-241.
- Gilmore D.J., (1990) "Expert Programming Knowledge : a strategic approach". In J.M Hoc, T.RG Green, R. Samurcay, D.J Gilmore (Eds) *"Psychology of programming"*, London, Academic Press, pp 223-224.
- Golberg A., Robson D. (1983) *"Smalltalk-80, The language and its implementation"* Addison-Wesley Publishing Company.
- Green TR. G., (1991) "What the psychologist's eye could be telling the software engineer's brain". In *NATO Workshop on User-centred requirements for software engineering environments"*, Bonas (France), 5-10 sept. 91, pp 323-335.
- Grenn T.R.G., Davies S.P., Gilmore D.. (1996) "Delivering cognitive psychology to HCI : the problems of common language and knowledge transfer". In *Interacting with Computers*, volume 8, n°1, pp 89-111.
- Grudin J. (1990) "The computer reaches out : the historical continuity of interface design". In *Conference of Computer Human Interaction* (CHI'90), ACM SIGGHI, pp 261-268
- Guegan J-C., Rosanvallon A., Troussier J-F.,(1987) "Nouvelles technologies et industries de process : d'un essai de définition a des examens de mise en oeuvre". In A. Silem (Ed) *"La diffusion des nouvelles technologies"*. Lyon, Centre Régional de Publication de Lyon-CNRS, pp 32-47
- Guidon R. (1990a) "Designing the Design Process : Exploiting Opportunistic Thoughts". In *Human Computer Interaction*, volume 5, pp 305-344.
- Guidon R. (1990b) "Knowledge exploited by experts during software system design. In *International Journal Man-Machines Studies*, 33, pp 297-304.
- Gulliksen J.; Sandbad B. (1995) "Domain-specific of User Interfaces". In *International Journal of Human Computer Interaction,* 7(2), pp 135-151.
- Habrias H. (1993) *"Méthodologie du logiciel. Introduction à la spécification"*. Paris, Masson.
- Hammer P., et Campy J.A. (1993) *"Le Reengineering"*. Paris, Dunod.
- Hammond J.K. (1989) *"Case-based planing : viewing planning as a memory task"* Ac. Press.
- Hammouche H. (1993) *"De la modélisation des tâches a la spécification d'interfaces utilisateurs"*. Rapport de Recherche n°1959, INRIA Rocquencourt.
- Harker S. (1987) "Rapid prototyping as a tool for user centred design". In G. Salvendy (Ed) *Cognitive Engineering in the design of Human Computer Interaction and Experts Systems.* Amsterdam, Elsevier Sciences Publications, pp 365-372.

- Harmon P. (1994) "Les systèmes client serveur. In *Génie Logiciel et Systèmes experts*, n°34, pp 4-27.
- Harvey L., Anderson J. (1996) "Tranfer of declarative knowledge in complex information-processing domains". In *Human Computer Interaction*, volume 11, pp 69-96.
- Hatchuel A. (1996) "Coopération et conception collective. Variétés et crises des rapports de prescription". In De Terssac G., Friedberg E. (Eds) "*Coopération et Conception*". Octares Edition, Toulouse, pp 101-122.
- Hayes-Roth B., & Hayes-Roth F., (1979), "A cognitive model of planing" In *Cognitive Science*, 3, pp 275-310.
- Hoc J-M, (1982) "L'étude psychologique de l'activité de programmation : une revue de la question". In *Techniques et Sciences informatiques*, (5), pp 383-392.
- Hoc J-M (1983) "Analysis of beginners problem solving strategy in programming" in T.R.G Green, S.J Payne, G.C van der Veer, (Eds) "*The psychology of computer use*" London, Academinc Press, pp 144-158.
- Hoc J-M (1987a) "*Psychologie cognitive de la planification*", Grenoble, PUG.
- Hoc J-M (1987b) "L'apprentissage de l'utilisation des dispositifs informatiques par analogie a des situations familières". In *La psychologie Française : Les langages informatiques dans l'enseignement.* T32-4, pp 217-226.
- Hoc J-M (1987c) "Prise de conscience et planification". *La psychologie Française : Les langages informatiques dans l'enseignement.* T32-4, pp 247-252.
- Hoc J.M. (1988) "L'aide aux activités de planification dans la conception de programmes". In *Le Travail Humain : Psychologie ergonomique de la programmation informatique*, Vol. 51, Tome 4, pp 321-333.
- Hoc J.M., Visser W. (1988) "Approche cognitive de la programmation informatique". In *Le travail humain*, Vol 51, N°4 pp 289-296.
- Hoc J-M., Amalberti R. (1995) "Diagnosis : some theoricals questions raised by applied research" . In *Cahier de Psychologie Cognitive*, 14 (1), pp 73-101.
- Holyoak K.J. , (1984) "Analogical thinking and human intelligence. In R.J Sternberg (Ed) "*Advances in the Psychology of human intelligence*", pp 199-230.
- Houriez B., Coupey P., Visser W. (1997) "Assistance à la construction et à la réutilisation de connaissances dans le cadre des activités de supervision". In *Colloque CNET-CNRS : Conception d'interfaces intelligentes et évolution des activités de télécommunication , Gif-sur-Yvette , 1-2 avril 1997.*
- Hoyos C.G., Gstalter H., Strube V., Zang B., (1987) "Software-design with the rapid prototyping approach : a survey and some empirical results". In G. Salvendy (Ed) "*Cognitive Engineering in the design of Human Computer Interaction and Experts Systems*". Amsterdam, Elsevier Sciences Publications, pp 329-340.
- Humphrey J. (1989) "Au delà de la critique du déterminisme technologique". In *Sociologie du travail*, n°2/89 pp 163-175.
- Iosif G.H, (1993), "Quelques aspects des relations entre modèle mental, représentation et modèle cognitif" in *Le travail humain*, T 56, n°4/93, pp 281-297.

- Jabes J (1991) "Changement et développement organisationnel". In N. Aubert, J.P Gruère, J. Jabes, H. Laroche, M. Sandra (Eds) "*Management : aspects humains et organisationnels*". Paris, PUF, pp 593-636
- Jeffroy F. (1993), "Connaître les caractéristiques de l'activité humaine pour concevoir des systèmes interactif", in *Génie Logiciel et Systèmes Experts*, n°33, pp 11-16.
- Johnson-Laird P.N (1989) "Mental Models". In M.I Pesnier (Ed), *Fondations of Cognitive Science*, Cambridge, MIT Press, pp 469-499.
- Kalagiakos P. (1993) "Reuse support environment of software ressources". In *Le génie logiciel et ses applications (6th Int Conf)*. 15-19 nov 1993. Le CNIT Paris La Défense, pp 295-304
- Karsenty L., Brézillon P. (1995) "Coopération Homme-Machine et explication". In *Le Travail humain*, Tome 58, n°4, pp 289-310.
- Kay A.C. (March 1993) "The early history of Smalltalk". In *ACM PLAN Notices*, vol 28, pp 1-54.
- Lange B.M., Moher T.G. (1989) "Some strategies of reuse in an object-oriented programming environment". In Bice K., Lewis C. (Eds), *CHI'89, Wings for the minds, Conference Proceedings*, Addison Wesley.
- Larsson T. et Vaino-Larsson A., (1991) "Software procedures as software users". In *NATO Workshop on User-centred requirements for software engineering environments,* Bonas (France), 5-10 sept. 91, pp 285-297.
- Lasfargues Y. (1995) "L'informatique repart 20 ans en arrière". In *Le monde informatique*, 15 Juin 1995.
- Lebahar J-C. (1992) "Quelques formes de planification significatives de l'activité de conception en design industriel". In *Le Travail Humain*, volume 55, n°4, pp 329-351.
- Lebahar J-C (1996a) "Cahier des charges réduites, tâche de conception en temps limité et commentaires critiques *a posteriori* : une approche pédagogique de la conception". In *Revue de Design/Recherche*, n°9, décembre 1996, pp 55-69
- Lebahar J-C. (1996b) "L'activité de simulation d'un dessinateur CAO dans une tâche de conception". In *Le Travail humain,* tome 59, n°3, pp 253-275.
- Léchevin J-P., Le Jolliff G., Lanoë D. (1994) "Vivre les nouvelles technologies". In *Cahier Travail et emploi*, Paris, La Documentation Française.
- Leplat J, Cuny X (1977) "*Introduction à la psychologie du travail*", Paris, PUF.
- Leplat J. (1985) "Les représentations fonctionnelles dans le travail" In *Psychologie Française*, n°30, pp 269-275.
- Leplat J. (1991) "Compétences en ergonomie". In M. Montmollin (Ed.), *Modèles en analyse du travail*, Bruxelles, Mardaga, pp 263-278.
- Maggi B. (1996) "Coopération et coordination : enjeux pour l'ergonomie". in J.C Sperandio (Ed) "*L'ergonomie face aux changements technologiques et organisationnels du travail humain*", Toulouse, Octares Edition, pp 11-25
- Malhotra A., Thomas J.C., Caroll J. M. et Miller L.A. (1980) "Cognitive processes in design". In *International Journal Man-Machines Studies*, 12, pp 119-140.

- Marinko M.J ; Jenry J.W., Zmud R..W. (1996) "An attributional explanation of indivudual resistance to the introduction of information technologies in the workplace". In *Behaviour & Information Technology*, Vo 15, n° 5, pp 313-330
- Mayer R.E. (1989) "Human nonadversary problem solving". In K. J. Gilhoody (Ed.) *Human and machine problem solving*, New York : Plenum.
- Mazoyer B., Salembier P. (1987) "Les principes techniques de receuil de données en ergonomie des logiciels". In C. Alezra, J. Christol , P. Falzon, B. Mazoyer, L. Pinsky, P. Salembier (Eds) *"L'ergonomie des logiciels : un atout pour la conception des systèmes informatiques"*, Paris, La Documentation Française, pp 22-26
- Menadier J-P (1991) *"Interface utilisateur : pour une informatique plus conviviale"* Paris, Dunod
- Michard A. (1993) "Maquettage et prototypage des interfaces". In J.C. Sperandio (Ed) *"L'ergonomie dans la conception des projets informatiques"*, Toulouse, Edition Octares, pp 163-213.
- Michel G. (1995) " *L'analogie dans l'apprentissage de la programmation pour des non-informaticiens*". Mémoire de DEA, Paris V.
- Midler C (1996) "Modèles gestionnaires et régulations économiques de la conception". In G. de Terssac, E. Friedberg (Eds) *"Coopération et Conception"*, Toulouse, Octares Edition pp 63-85
- Morais A., Visser W. (1985) *"Etude exploratoire de la programmation d'automates industriels chez les élèves de l'enseignement technique"*, Rapport de Recherche n° 404, INRIA Rocquencourt.
- Morais A. Visser W. (1987) "Programmation d'automates industriels : adaptation par des débutants d'une méthode de spécification de procédures automatisées". In *La psychologie Française : Les langages informatiques dans l'enseignement.* T32-4, pp 253-260.
- Montmollin M. (de) (1984), *"L'intelligence de la tâche"*, Berne, P. Lang.
- Nanard J. (Dec 1990) *"La manipulation directe en interface Homme/machine"*. Thèse d'état en informatique. Université de Montpellier,.
- Nanard M. (1991) "Conception par objets d'Interfaces Homme-Ordinateur". In *5Th Autumn school at Campus Thompson*,.
- Newell A.; Shaw J.C et Simon H.A (1959) "Report on a general problem-solvong program". In *Proceedings of ICIP*, pp 256-264.
- Nguyen-Xuan A.(1987) "Apprentissage par l'action d'un domaine de connaissance et apprentissage par l'action du fonctionnement d'un dispositif de commande". In *La psychologie Française : Les langages informatiques dans l'enseignement.* T32-4, pp 237-246.
- Nguyen-Xuan A. (1990) "Le raisonnement par analogie". In J-F. Richard, C. Bonnet, R. Ghiglione (Eds) *"Traité de psychologie cognitive : le traitement de l'information symbolique" (vol.2)*. Paris, Dunod, pp 147-155.
- Norman D.A (1984) "Stages and levels in human-machine interaction". In *International Journal of Man-Machine Studies*, 21, pp 365-375.

- Norman D.A (1991) "Cognitive artefacts". In *Designing Interaction : psychology at the Human-Computer Interface*, Ed. J.M. Carroll, Cambridge University Press.
- Norman D.A. (1983) "Somme observations on mental models. In D. Gentner, A.L. Stevens *(Eds) "Mental Models"*. Hillsdale, Lawrence Erlbaum Associates, pp 15-34.
- Ombredame A, Faverge J-M. (1955) "*L'analyse du travail*", Paris, PUF.
- Owen, D. (1987) "Direct Manipulation and Procedural Reasoning". In G. Salvendy (Ed) "*Cognitive Engineering in the design of Human Computer Interaction and Experts Systems*". Amsterdam, Elsevier Sciences Publications, pp 349-356.
- Pair (1988) "Langages et méthodes de programmation". In *Psychologie ergonomique de la programmation informatique*, Le Travail Humain, Volume 51, n°4, pp 297-307.
- Palanque P.n Bastide R. (1995) "Spécifications formelles pour l'ingénierie des interfaces homme/machine". In *Techniques et sciences informatiques*, vol 14, n°4/95, pp 473-500.
- Papert S. (1981) "*Jaillissement de l'esprit. Ordinateur et apprentissage*". Paris, Collection Champs, Flamarion..
- Papert S., Solomon C. (1986) "Twenty Things to do with a computer". In E. Soloway, J.C. Spoher (Eds), *Studying the novice programmer.* Addison-Wesley Editor.
- Pennington N., Lee A. et Rehder B. (1995) "Cognitive activities and levels of abstraction in procedural and object-oriented design"; In *Human Computer Interaction*, volume 10, pp 171-226.
- Perron L. (1994) "*La réutilisation de connaissances "vers une aide à la réutilisation de cas..."*. Rapport de Recherche n°2415, INRIA Rocquencourt.
- Perron L. (1995). "La réutilisation de cas : une problématique commune entre I.A. et ergonomie cognitive... des points de vue différents". In *Actes du colloque JAVA'95 (Journées Acquisition, Validation, Apprentissage)*, Grenoble, INRIA, pp. 199-212.
- Perrot J.F. (Nov 1994) "*Langages à objets*", LAFORIA 94/25, Institut BLAISE PASCAL, Université Paris VI-VII.
- Petre M. (1990) "Experts Programmers and Programming Languages". In J.M Hoc, T.R.G Green, R. Samurcay, D.J Gilmore (Eds) "*Psychology og programming*", Academic Press, pp 103-116.
- Pluchart (1997) "L'externalisation des activités immatérielles des entreprises". In *Humanisme et Entreprise*, n°22, T 5, pp 61-740.
- Prigent V. (1995) "L'appropriation d'une nouvelle technique de cmmunication dans l'entreprise". In *Humanisme et Entreprise*, n°95, pp 81-102.
- Rabardel P. (1995) *"Les hommes et les technologies : approche cognitive des instruments contemporains"*. Paris, Armand Colin.
- Rabin S. (1995) "Transitioning Information Systems cobol developers into objet cobol technicians". In *Object Magazine* (janvier 1995), pp 71-76.
- Rasmussen J. (1986) "*Information processing and human-machine interaction*", North Holland, New-York..

- Rauterberg M. and Strohm O. (1992) "Work organization and software development" In *Annual Review of Automatic Programming* Vol. 16, pp 121-128.
- Rauterberg M., Strohm O. et Kirsch C. (1995) "Benefits of user-oriented software development based on an iterative cyclic process model for simultaneous engineering". In *International Journal of Industrial Ergonomics,* 16(4-6), pp 391-410.
- Rauterberg M. (1996)."How to measure the ergonomic quality of user interfaces in a task independent way". In A. Mital, H. Krueger, S. Kumar, M. Menozzi, J.E Fernandez (Eds.), "*Advances in Occupational Ergonomics and Safety*", Cincinnati: International Society for Occupational Ergonomics and Safety, pp 154-157.
- Repenning A., Summer T. (1995) "*Agentsheets : a medieum for creating domain-oriented visual languages*". IEEE Computer.
- Richard J.F. (1983) "*Logique de fonctionnement et logique d'utilisation*". Rapport de recherche n°202, INRIA Rocquencourt.
- Richard J-F (1990) "*Les activités mentales : comprendre, raisonner, trouver des solutions*". Paris, Colin.
- Rist (1991) "Knowledge creation and retrieval in program design : a comparaison of novice and intermediate student programmers". In *Human-Computer Interaction,* 6, pp 1-46.
- Rist R.S. (1990) "Variability in program design : the interactioin of process with knowledge". In *International Journal of Man-Machine Studies*, 33, pp 305-322.
- Rogalski J., Samurçay R., Hoc J.M (1988) "L'apprentissage des méthodes de programmation comme méthodes de résolution de problème" In *le travail humain* Vol 51, N°4, pp 309-320.
- Rosson M.B., Gold E., (1989), "Problem-Solution Mapping in Object-Oriented Design". In *OOPSLA '89 Proceedings*, pp 7-10.
- Rosson M.B, Alpert S.M. (1991) "*The cognitive consequences of object-oriented design*", RC 14191, IBM Research Report ,Watson Research Center.
- Russel A.J., Galer M.D. (1987) "Designing human factors design aids for designers". In G. Salvendy (Ed) "*Cognitive Engineering in the design of Human Computer Interaction and Experts Systems*". Amsterdam, Elsevier Sciences Publications, pp 289-296.
- Sacerdoti E.D. (1974) "Planning in a Hierarchy of asbstractions spaces". In *Artificial Intelligence*, 5, pp 115-135.
- Samurçay R. (1987) "Plans et schémas de programme" . In *La psychologie Française : Les langages informatiques dans l'enseignement.* T32-4, pp 261-266.
- Schnaider Hufschmidt M. (1991) "Designing user interfaces by direct composition prototyping appearences and behavior of user interfaces". In *NATO Workshop on User-centred requirements for software engineering environments,* Bonas (France), 5-10 sept. 91, pp 235-253.
- Scholtz J., Wiedenbeck S., (1990) "Learning to program in another language". In D. Diaper, D. Gilmore, G. Cockton, B. Shackel (Eds), "*Human computer interaction : Proceedings of Interact '90*", North Holland, pp 925-930.

- Scholtz J. (1991) "The effect of the mental representation of programming knowledge on transfert". In In *NATO Workshop on User-centred requirements for software engineering environments,* Bonas (France), 5-10 sept. 91, pp 119-127.
- Scholtz J., Wiedenbeck S., (1993) "An analysis of novi programmers learning a second language". In C.R Cook, J. Scholtz, Spohrer (Eds), "*Empirical studies of programmers : Fifth workshop*", Ablex, pp 187-205.
- Sebillote S. et Bisseret A. (1986) "*La conception des scénarios interactifs*". Rapport de Recherche n°537. INRIA Rocquencourt.
- Sebillote S. (1988) "Hierarchical planing as method for task analysis : the exemple of office task analysis". In *Behavior and Information Technology* , 7, (3) pp 275-293.
- Sebillote S. (1991) "*Décrire des tâches selon les objectifs des opérateurs : de l'interview à la formalisation*". Rapport Technique n°125, INRIA Rocquencourt.
- Sebillote S. (1993) "Schémas d'actions acquis par l'expérience dans les représentations mentales des opérateurs : leurs utilisations et la construction de nouveaux schémas". In A. Weill-Fassina , P. Rabardel, P. Dubois (Eds) "*Représentations pour l'action*". Toulouse, Edition Octares.
- Senach B. (1990) "*Evaluation ergonomique des interfaces homme-machine : une revue de littérature*". Rapport de Recherche n°1180, INRIA Rocquencourt.
- Soberman M. (1992) "*Génie logiciel en informatique de gestion*". Paris, Eyrolles.
- Soloway E. et Ehlich K. (1984) "Empirical Studies of programming knowledge" In *IEEE Transactions on Software Enginneering*, SE-10 (5), pp 595-609.
- Sonnentag S; (1993) "What is so special about exceptional Software professionals ?" In T. Grechening, M. Tscheligi (Eds), "*Human Computer Interaction*", Vienna Conference, VHCI'93, Berlin Springer, pp 353-368.
- Sperandio J-C. (1987) "Introduction à l'ergonomie des logiciels". In C. Alezra, J. Christol , P. Falzon, B. Mazoyer, L. Pinsky, P. Salembier (Eds) "*L'ergonomie des logiciels : un atout pour la conception des systèmes informatiques*", Paris, La Documentation Française, pp15-21
- Sperandio J-C (1988) "*L'ergonomie du travail mental*". Paris, Masson.
- Sperandio J-C (1995) "Modéliser le savoir et les activités opératoires par les formalismes de l'informatique, est-ce pertinent pour l'ergonomie". In *Performances Humaines et Techniques : Séminaire DESUP/DESS de Paris*, Toulouse, Octares Edition, pp 17-24.
- Stacy W., Mancmillian J. (1995) "Cognitive biais in Softaware Engineering". In *Communications of the ACM*, Mai 1995, Volume 38, n°5, 58-36.
- Straggers N., Norcio A.F. (1993) "Mental models : concepts for Human Computer Interaction Research". In *International Journal Man-Machines Studies*, 38, pp 587-605.
- Streitz N.A (1987) "Cognitive compatibility as a central issue in HCI : theorical framework and empirical findings". In G. Salvendy (Ed) *Cognitive Engineering in the design of Human Computer Interaction and Experts Systems.* Amsterdam, Elsevier Sciences Publications, pp 75-82.

- Strüng J. (1991) "Designing in working process. What programmers do beside programming ?". In *NATO Workshop on User-centred requirements for software engineering environments*, Bonas (France), 5-10 sept. 91, pp 81-91.
- Suchman L. (1987) "*Plans and situated actions, The problem of human/machine communication*". Cambridge ; Cambridge University Press.
- Sybord C. (1995) "Informaticiens : quel est votre devenir". In *Actes du 10° colloque Européen en Informatique et Société (CREIS), "Responsabilités sociales et formation des acteurs de l'informatisation"*, 5-7 juillet 1995, Belgique, Namur, pp 49-56
- Teasley B., Leventhal L.M, Rholman S. (1993) "Positive test biais in software testing by professionals : wath's right and what's wrong". In C.R. Cook, J.C. Scholtz, C. Spoher (Eds), "*Empirical Studies of Programmers : Fifth Workshop*", Eds Ablex, Norwood.
- Teiger C. (1993/3) "L'approche ergonomique : du travail humain à l'activité des hommes et des femmes au travail". In "*Education permanent :comprendre le travail*", n°116, pp 71-96.
- Terssac G. (de), Friedberg E. (1996) "*Coopération et Conception*", Toulouse, Octares Edition.
- Theureau J., Jeffroy F. (1994) "*Ergonomie des situations informatisées : la conception centrée sur le cours d'action des utilitsateurs*". Toulouse, Octares Edition.
- Traunmuler R., (1988), "Conceptuals models for designing Informations Systems". In G.C. Van Der Veer, T.R.G. Green, J.M. Hoc & D.M. Murray (Eds) *"Working with Computers : Theory versus Outcome"*, London , Academinc Press, pp 123-144.
- Tulving E., Thomson D.M. (1973) "Encoding specificity and retriaval processes in episodic momory". In *Psychological Review*, 80, pp 352-373.
- Van zuylen (1991) "Views and representations for reverse engineering". In *NATO Workshop on "User-centred requirements for software engineering environments"*, Bonas (France), 5-10 sept. 91, pp 41-57.
- Varela F. J. (1990) "*Connaître les sciences cognitives : tendances et perspectives*", Paris, Seuil.
- Visser W. (1988) "L'activité de comparaison de représentations dans la mise au point de programmes". In *Le Travail humain*, tome 51, n°4, pp 351-362.
- Visser W., Morrais A. (1988) "Concurrent use of differents techniques for gathering data on the programming activity", Rapport de Recherche n°939, INRIA Rocquencourt.
- Visser W., Hoc J-M. (1990) "Expert software design strategies. In J.M Hoc, T.R.G Green, R. Samurcay, D.J Gilmore(Eds) *Psychology of programming,* Academic Press, pp 235-250.
- Visser W. (1991) "Planning and organization in expert design activities". In *NATO Workshop on User-centred requirements for software engineering environments,* Bonas (France), 5-10 sept. 91.
- Visser W., (1992a), "*Design organization: there is more to expert knowledge than is dreamed of in the planner's philosophy*", Rapport de Recherche n°1765, INRIA Rocquencourt.

- Visser W. (1992b) "Designers' activities examined at three levels : organization, strategies and problem solving . In *Knowledge-Based Systems.* 5(1), pp 92-104.
- Visser W. (1995a) "Coût cognitif : comme concept explicatif de la différence entre un plan d'activité et l'organisation effective de cette activité". In *Atelier de Conjoncture "La charge cognitive"*. Département de recherche de la Société Française de Psychologie, Montepellier, 30 Juin-1 Juillet 1995, pp 113-125
- Visser W. (1995b) "Reuse ok Knowledge : empirical studies". In M. Veloso, A. Aamodt (Eds.), *"Case-based reasoning. Research and development"*. First International Conference, ICCBR-95. Sesimbra, Portugal, Berlin : Springer, pp. 335-346.
- Visser W. (1995c) "Use of episodic knowledge and information in design problem solving". In *Design Studies*, volume 16, pp 171-187.
- Wallace K. ; Hales C. (1987) "Détailed analysis of an engineering design project". In *Proceedings of the International Conference of Engineering Design*, Boston.
- Wertz H. (1993) "L'intelligence des environnements de programmation". In J-C Sperandio (Ed) *"L'ergonomie dans la conception des projets informatiques"*, Toulouse, Edition Octares, pp 163-213.
- Weil-Barais A. (1991) "Résolution du problème". In J-P. Rossi (Ed) *"la recherche : Domaines et méthodes"*, Paris, Dunod, pp 105-158 .
- Weil-Barais A. (1994) "Comment l'homme apprend-il, raisonne-t-il, juge-t-il et résout-il des problèmes ?". In A. Weil-Barais (Ed) *"L'homme cognitif"*. Paris, Puf, pp 413-545
- Winograd T., Flores F. (1989) *"L'intelligence artificielle en question"*, Paris, Puf.
- Winograd T. (1993) "Heidegger et la conception des systèmes informatiques". In *Intellectica*, 93/2, pp 51-78.
- Winograd T. (1995) "From programming environments to environments for designing". In *Communications of the ACM*, Mai 1995, Volume 38, n°5, pp 45-52.
- Wisner A (1987) "Préface". In C. Alezra, J. Christol , P. Falzon, B. Mazoyer, L. Pinsky, P. Salembier (Eds) *"L'ergonomie des logiciels : un atout pour la conception des systèmes informatiques"*. Paris, La Documentation Française, pp 7-11.
- Wisner A. (1996) "Du contenu à la forme de la cognition située (aspects psychologiques de l'anthropotechnologie)". In R . Patesson (Ed) *"Intervenir par l'ergonomie"*, XXXI^{e} congrès de la SELF, Bruxelles, Creatic, Vol .2, pp 94-99.

ANNEXES

ANNEXE I

Principes et modalités d'intervention de la Recherche-Action

Notre intervention s'inscrit dans le courant de la "recherche-action", secteur de recherche développé depuis la première moitié du XX^{e} siècle par des auteurs comme Lewin, Moreno et Mayo en Amérique du Nord, Jacques et Tavistock en Europe (Dubost, 1987). Cette approche répond à un certain nombre de critères :

- il s'agit d'une expérience s'inscrivant dans le *mode réel*, dans une histoire concrète : les actes posés par les agents prennent le caractère d'événements pour tous ceux qui sont concernés ; de ce point de vue, chaque opération à un caractère irréversible, contrairement à ce qui se passe en général dans les recherches de laboratoire.
- Cette expérience est engagée à une échelle restreinte : cette limitation peut être le résultat du caractère local (limitation à un ou plusieurs cas) ou de l'application d'un principe d'échantillonnage. Mais cette réduction d'échelle n'affecte par de manière radicale la durée du processus, contrairement à beaucoup de recherches en laboratoire.
- En tant qu'actions délibérées visant un changement effectif au niveau des zones concernées, elle se définit par des buts qui peuvent être fixés soit par les initiateurs du projet et par les instances centrales du pouvoir, soit par l'ensemble ou un sous-ensemble d'individus et de groupes engagés dans le processus, soit encore par un processus de négociation entre les différents acteurs concernés.
- Elle est conçue dès son engagement pour permettre d'en dégager des enseignements susceptibles de généralisation, pour guider des actions ultérieures ou mettre en évidence des principes ou des lois ; elle tente donc de disposer de certaines capacités d'anticipation en liaison avec un projet plus général qui l'englobe, se situe à une autre échelle spatiale et temporelle et dont certain aspects peuvent être modifiés à terme par ses résultats.

- Elle doit donc accepter certaines disciplines permettant l'observation, la récolte d'informations, l'enregistrement de traces dont le traitement conditionne la production de résultats, le contrôle et l'évaluation des effets, etc. et/ou permettant l'exploration de divers aspects, l'interprétation, la prise de conscience…

En ce qui concerne ses modalités d'intervention, la recherche action tente d'éclairer les relations qui s'établissent entre : le *système* où la demande surgit (dans notre cas : *le service DABFI*), les *objets* sur lesquels se centre le travail (*les environnements de conception*), les *acteurs* qui y participent ou en sont écartés (*les informaticiens*), les *agents* dont la collaboration est demandée (*responsables hiérarchiques et structures d'accompagnement et de formation*), les *facteurs* qui ont généré le problème et la demande de consultation (*le transfert technologique*), les *effets* produits par le processus (*transferts d'apprentissage et déterminisme technologique*).

ANNEXE II

Situation de l'entreprise où s'est déroulée l'étude

Cette recherche a été effectuée à Informatique-CDC, filiale du groupe "Caisse des Dépôts et Consignation" (Cf. Figure 72). Les informaticiens conçoivent, développent et gèrent les systèmes d'information de ce groupe financier. L'organisation d'Informatique-CDC se compose de 4 directions fédérales et en 6 GIE[63] localisés entre Angers, Bordeaux, Paris, Arcueil et Bagneux.

Chaque GIE travaille pour une branche spécifique du groupe Caisse des Dépôts :

- DABFI[64] a pour client la Direction des Affaires Bancaires et Financières (DABF),
- DFEI : la Direction des Fonds d'Epargne (DFE) et la direction aux Collectivités Locales (DCL),
- GIP : CNP assurances,
- GIRET : la Branche Caisses de Retraites (BCR),
- GIS : le secrétariat général,
- et CLF-SI : Crédit Local de France qui est devenu un partenaire du groupe CDC depuis sa privatisation.

Les directions fédérales —Direction Technique, la Direction des Moyens d'Exploitation (DMCE), la Direction des Ressources Humaines (DRH) et la Direction Financière (DF)— assurent les missions d'intérêts communs à Informatique-CDC (mise à disposition de ressources et moyens partagés, missions d'audit et de diagnostic...)

[63] Groupement d'Intérêts Economiques
[64] Dabfi est le service où s'est déroulé notre recherche

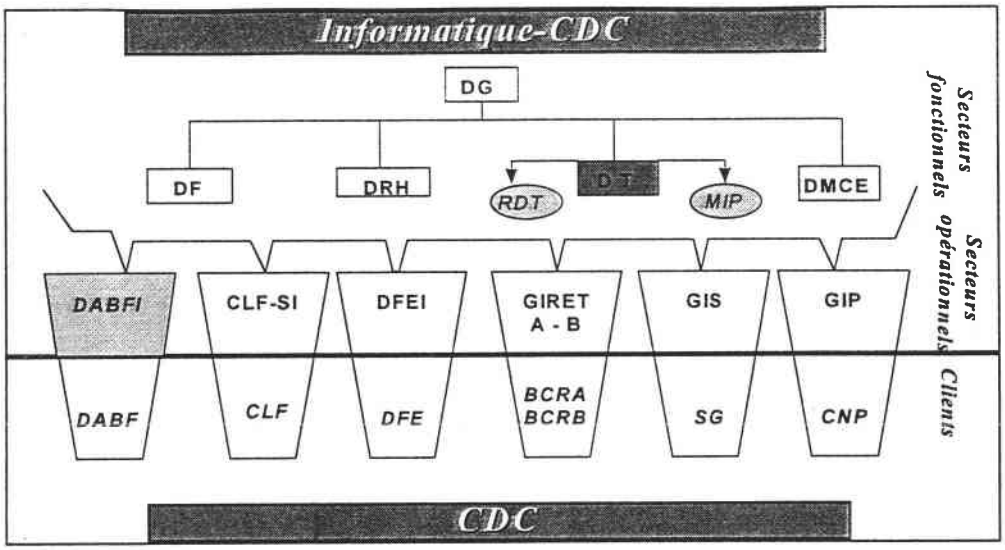


Figure 72
Organigramme fonctionnel de l'entreprise

ANNEXE III

LES MÉTHODES DE RECUEIL DE DONNÉES ET LES PROBLÈMES POSÉS PAR LES TECHNIQUES DE VERBALISATION

S'appuyant sur la monographie et l'analyse de protocoles verbaux (Bainbridge, 1990), différentes techniques de recueil ont été utilisées dans cette recherche :

1. **Des entretiens exploratoires** : ces entretiens exploratoires, menés de manière semi-directive, ont pour but d'obtenir des informations générales sur l'organisation de l'activité, ou plus précisément de la tâche. C'est en effet une situation de recueil où les informaticiens interrogés fournissent une certaine représentation du travail qu'ils pensent réaliser ou projettent de réaliser. Elle ne peut donc correspondre à ce qui se fait réellement dans l'activité et à la manière dont cette activité se déroule concrètement dans un contexte particulier de travail. Des recueils supplémentaires sont donc nécessaires pour cerner la réalité et la complexité du travail , en particulier l'observation et les techniques de verbalisation.

   Les thèmes abordés dans ces entretiens exploratoires portent sur les caractéristiques de l'activité, sur la description du déroulement de l'activité, sur les conditions d'utilisation de l'outil, sur les difficultés rencontrées durant la tâche de maquettage, sur la nature des informations traitées ou recherchées, sur le genre de contraintes à gérer, sur le type d'assistance requis (vers qui et pourquoi), sur les situations problèmes et sur les stratégies déployées pour les résoudre, sur la répartition des tâches entre les informaticiens, et la nature des relations dans l'équipe.

   Des **entretiens complémentaires et ciblés** sont également réalisés à la suite des phases d'observation de l'activité. Leur finalité est d'obtenir des explications ou des précisions sur certaines conduites remarquées durant l'activité de maquettage.

2. **L'observation de l'activité** : à partir de éléments recueillis lors des entretiens exploratoires, nous nous centrons sur un certain nombre d'éléments plus particuliers à l'activité de maquettage. Les aspects retenus sont ceux qui apparaissent, sinon comme des facteurs explicatifs, du moins comme permettant de donner un sens aux comportements de l'informaticien. Le but est de collecter toutes les informations relatives à l'activité réelle de l'individu : difficultés rencontrées, informations recherchées, utilisées ou ignorées, nature et déroulement des séquences opératoires, nature et fréquence des interactions homme-machine et homme-homme, outils et documents utilisés, types d'erreurs commises, etc.

   Toute ces données participent finalement à la définition et à la compréhension de l'activité réelle de l'opérateur (Mazoyer et Salembier, 1987).

3. **L'analyse de traces et des documents** portant sur l'activité en cours : cela concerne tous les supports (papier et électronique) que l'informaticien utilise comme une aide ponctuelle ou systématique au cours de son travail. Ces documents peuvent avoir été rédigés par l'informaticien lui-même durant son activité (brouillon), sur la base de son expérience (aide-mémoire), ou bien avoir été édités par des cellules spécialisées d'assistance et de méthodes (manuel de procédures par exemple) (Visser et Morrais, 1988).

4. Des **verbalisations simultanées** à la conception de l'interface : elles sont des indicateurs de l'activité mentale en temps réel. De manière pratique, il s'agit pour le développeur de penser à "*haute voix*" durant la conception de sa maquette. La verbalisation est en effet une situation où le contenu de la mémoire de travail s'exprime assez clairement. Et c'est à partir de ces éléments que des inférences peuvent être faites sur les représentations mentales et les procédures mobilisées par le concepteur lors de la résolution de problèmes informatiques (Caverni et Bastien, 1988).

5. Le **tri-conceptuel** (Gammack et Young, 1984) : il vise à la verbalisation de l'organisation mnémonique de l'informaticien. Ce dernier est prié de formaliser tous les concepts pertinents pour son activité. Puis, le psychologue

ergonome rédige une fiche d'information sur chaque concept (un post-it pour un concept) et demande à l'informaticien de les organiser en groupes et en sous-groupes, et de justifier de ses choix de classement. Cette méthode permet d'appréhender, et la nature des connaissances employées au cours de l'activité (techniques, fonctionnelles, stratégiques...), et leur structuration mnémonique.

Concernant ces 3 dernières techniques de verbalisation, de nombreuses critiques ont été soulevées par les chercheurs sur le statut de la verbalisation en tant que témoignage des processus cognitifs en oeuvre. La verbalisation suppose d'abord un minimum de capacité d'abstraction et de formalisation des expériences, des sensations éprouvées..., et suppose surtout la possibilité de « *mettre à mots* » ; c'est-à-dire d'avoir à sa disposition "*les mots pour le dire*" (Teiger, 1993). Ce qui veut dire que le déficit de vocabulaire rendrait impossible l'explicitation des conduites mentales. Par ailleurs, la connaissance serait "incorporée" à l'activité, et de ce fait inaccessible à soi comme aux autres, car non verbalisées. Aussi, les connaissances ne pourraient être ni verbalisables, ni formalisables, et ni conscientisables. Enfin, d'un point de vue méthodologique, on critique le fait de demander à l'individu de se concentrer sur une tâche de verbalisation, alors que, dans le même temps, toute son attention est retenue dans l'exécution d'une autre activité. On crée ainsi les conditions d'une surcharge de travail par ajout d'une tâche complémentaire à l'activité principale. Cette surimpression de tâches fait qu'on ne disposerait que d'une couche très superficielle des processus mis en oeuvre.

Malgré toutes ces critiques, nous pensons que ces techniques restent le seul moyen pour rendre compte ou tout au moins, pour s'approcher des mécanismes cognitifs activés durant le travail. En cela, nous rejoignons le point de vue théorique sur le cours d'action formulé par Pinsky et qui le définit comme « *l'activité d'un acteur déterminé, engagé activement dans un environnement physique et social déterminé et appartenant à une culture déterminée, activité qui est significative pour ce dernier, c'est-à-dire montrable, racontable et commentable par lui à tout instant de son déroulement à un observateur-interlocuteur* » (Cité par Theureau et Jeffroy, 1994, p 19)

ANNEXE IV

GRILLE D'ÉVALUATION ERGONOMIQUE DE PACBASE ET DE NSDK

Cette grille d'évaluation ergonomique a été élaboré à partir des critères ergonomiques définis par Bastien et Scapin (1993)

CARACTÉRISTIQUES TECHNIQUES DES ENVIRONNEMENTS		
	PACBASE	**NSDK**
– Système d'exploitation	*MVS (site-central)*	***Windows***
– Compatibilité ou connexion avec d'autres logiciels	*Pacdesign + Pacbench*	***Compatible avec tous les logiciels et progiciels de l'environnement Windows***
– Extensions possibles	*Terminal + DB2 (base de données)*	***Sybase***
Matériel de base :		
– Type de matériel	*Moniteur 3270*	***PC***
– Environnement technique	*site-central/ MVS/ TSO / Interface Alpha-numérique/ Affichage séquentiel.*	***client serveur***

1. GUIDAGE

Objectif général:
Evaluer l'ensemble des moyens mis en oeuvre pour conseiller, orienter, informer et conduire l'informaticien lors de ses interactions avec l'informaticien.

1.1 INCITATION

Recouvre l'ensemble des moyens mis en oeuvre pour amener l'informaticien à effectuer des actions spécifiques (entrées de données ou autres)

	PACBASE	NSDK
Existe-t-il un guidage des entrées des données indiquant le format adéquat ou les valeurs acceptables?	**Non**	**Non**
Un indice renseigne-t-il sur la longueur autorisée des entrées dans un champ?	**Non**	**Non**
Existe-t-il un titre pour chaque fenêtre ?	**Oui**	**Non**
Message d'erreur guide-t-il l'informaticien pour qu'il réalise l'objectif souhaité ?	**Non** *Par exemple: rien n'est dit sur le fait que des rubriques réutilisables soient disponibles dans la table ou dans le dictionnaire. De même, aucun ne message ne signale chevauchement des rubriques sur l'écran après le maquettage.*	**Oui**
AIDE		
* *en ligne ?*	**Oui** *Cette aide donne la définition des abréviations*	**Oui**
* *explicite, concise, claire ?*	**Peu explicite, et incomplète :** *L'informaticien doit se référer aux manuels d'aide*	**Oui**
* *spécialisée par Novice / Expert ?*	**Non**	**Non**
LIBELLÉS DES MENUS	**Non Pertinent (Non Pertinent)** *Il n'existe pas de menus sur les écrans alpha-numériques. En revanche, Pacbase propose un écran général où sont recensées les principales commandes disponibles*	
* *Menus classés par types de tâches ?*	*Commandes classées dans l'ordre logique des tâches: d'abord générale au développement , puis particulière à la sous tâche choisie (programmation ou maquettage). Mais cet écran de commandes n'est pas accessible au cours de l'utilisation d'un autre écran de Pacbase.*	**Oui** *Menu File, menu Projet*
* *Relation explicite entre les actions et les états (annoncée par un message) ?*	**Non**	**Oui**
* *Elimination des pistes inutiles ?*	**Non**	**Oui**

* *Actualisation de la représentation du menu en fonction du nouveau contexte de travail ?*	**Non**	**Oui** *Par exemple, lors de la tâche programmation des événements, apparition de nouvelles options dans le menu*
* *Indique-t-il clairement la différence entre un état initial et un état final ?*	**Non**	**Non**
* *Nombre d'options par menu ?*	**17 commandes générales pour la composition de l'écran de maquettage**	**9 en moyenne** *(13 aux plus, 3 au moins)*
* *Longueur des menus ?*	**NP**	**10 menus**
* *Reliés à d'autres menus ?*	**Oui** *Fonctionne de proche en proche : un choix de commandes appel un autre écran de commandes...*	**Non**
Boîtes de dialogue ?	**NP (n'existe pas)**	
* *Cohérence entre les différentes boîtes de dialogue (dispositifs de saisie)*	**NP**	**Oui**
* *Saisie des entrées dans un champ par défaut*	**NP**	**Oui**
* *Affichage des unités de mesure des données à saisir*	**NP**	**Non**
* *Pertinence des dispositifs de saisie ?*	**NP**	**Oui**

1.2. GROUPEMENT / DISTINCTION ENTRE ITEMS

Cela concerne l'organisation visuelle des items d'information les uns par rapport aux autres.

1.2.1 Groupement / Distinction par la localisation

Indique l'appartenance ou la non appartenance à une même classe ou encore dans le but de montrer la distinction entre différentes classes.

	PACBASE	NSDK
Les options des menus sont-elles groupées en fonction des objets sur lesquels elles s'appliquent ?	**Oui** ***Ecran de commandes de navigation dans Pacbase / Ecran de commandes de programmation / Ecran de commandes d'accès à la bibliothèque***	**Oui**
L'organisation des options est-elle logique ?	**Oui** *Le regroupement des commandes est regroupée en fonction de l'ordre logique des tâches à effectuer par l'informaticien*	**Oui** *Les premières option proposées sont les plus utilisées*
Le regroupement logique d'information est-il favorisé en :		
* Aire principale de travail	**Oui** *L'écran constitue la seule aire de travail disponible, et propose une zone pour la saisie des commandes*	**Oui** *Fenêtre de composition des fenêtres*
* Aire d'information sur les opérations en cours:	**Non**	**Oui** *Zone en bas des fenêtres*
* Aire des messages d'erreurs, des confirmations	**Oui** *Zone en bas de l'écran*	**Oui** *Fenêtre de message*

* Aires d'icônes et d'activités multiples	**Non Pertinent**	**Non**
* Aire d'assistance à l'opérateur	**Non**	**Oui** *Fenêtre d'assistance*
* Aire titre à l'écran	**Oui**	**Oui**
* Ces aires sont-elles figées / mobiles ?	**Figés**	**Mobiles**
Concernant le séquencement des écrans:		
* Le nombre de fenêtres pouvant se recouvrir est-il important ?	**N.P** *Pas de recouvrement possible : Monofenêtrage*	**Non** *3 à 4 fenêtre et boites de dialogue au plus peuvent se recouvrir*
* Existe-t-il des indices mnémotechniques diminuant la mémorisation entre les écrans ?	**Non**	**Non**
* Pages marquées ?	**Oui** *(par exemple, deux écrans pour la grille de maquettage : -CE1 et -CE2*	**Non**
* Recouvrement des fenêtres est-il logique? (linéarité)	**Oui**	**Oui**

1.2.2 Groupement / Distinction par le format ?
(Caractéristiques graphiques)

	PACBASE	NSDK
Existe-t-il une distinction visuelle entre des aires aux fonctions différentes ?	**Oui** *Aire d'information (bleue) ≠ aire de saisie (verte) ≠ aire d'erreur (rouge)*	**Oui**
Idem pour commandes ?	**Oui**	**Non**
Idem pour messages (d'erreur, confirmation, annulation) ?	**Oui** *Rouge*	**Oui** ***Par exemple, une petite icône*** *"Attention" signale les erreurs*
Les commandes par défauts sont-elles distinguées visuellement pour une validation rapide (ex: "OK") ?	**NP**	**Oui**
Une ligne de séparation existe-t-elle entre un ensemble d'options liées à une fonctionnalité et un autre ensemble d'options dédiées à une autre fonctionnalité ?	**Non**	**Oui**
La couleur est-elle employée pour symboliser les séquencements temporels, logiques et/ou physiques ?	**Non**	**Non**

1.3. FEED-BACK IMMEDIAT

(Concerne les réponses de l'ordinateur consécutives aux actions de l'informaticien)

	PACBASE	NSDK
Les entrées effectuées par l'informaticien sont-elles toujours visibles?	**Oui**	**Oui**
La lenteur du traitement est-il représenté par un item? (montre, message d'attente)	**Oui** ***Par une montre***	**Oui** *Par un sablier*
Les temps de réponse sont ils stables ?	**Non** *Dépend de l'encombrement du site-central*	**Non** *Dépend du nombre d'applications ouvertes simultanément sur l'interface*
Suite à l'interruption d'un traitement par l'informaticien, le système fournit-il un message indiquant le retour à l'état précédant?	**Non**	**Non**
Le système demande-t-il une confirmation systématique avant une action irréversible ?	**Non**	**Oui**
Un message indique-t-il que la commande a été exécutée ?	**Oui** *Message avec nombre de lignes crées, supprimées*	**Non**
Un message d'erreur informe-t-il l'informaticien sur la mauvaise action accomplie ?	**Oui**	**Oui**
* est-il clair, compréhensible, concis ?	**Concis : oui - Clair : Non.** *Par exemple : "E" pour déterminer une ligne d'erreur*	**Oui**
* est il neutre ?	**Oui**	**Oui**

1.4. LISIBILITE

(Caractère lexical de présentation pouvant entraîner ou faciliter la lecture de ces informations)

	PACBASE	NSDK
Caractères à l'écran sont ils lisibles ?	**Oui**	**Oui** *(sauf pour quelques caractères en reliefs)*
Luminance des caractères par trop forte?	**Non**	**Non**
Contraste caractère/fond ne choque pas?	**Non**	**Non**
Dimension des lettres homogènes ?	**Oui**	**Oui**
Découpage des chaînes de caractères est-il explicite et logique ?	**Non** *Nécessite de connaître le langage: abréviations absconses (par exemple: "RH " pour "Répétition Horizontale")*	**Oui**
Existe-t-il une unique police de caractères dans l'ensemble des messages ?	**Oui**	**Oui**
Titre centré ?	**Non**	**Oui**
Label en majuscule ?	**Oui**	**Oui**
Curseur repérable ?	**Oui** *(mire + couleur)*	**Oui**
La composition des messages est-elle faite de quelques lignes courtes ou de plusieurs lignes longues	**Une seule ligne longue**	**Quelques lignes courtes**
Les césures de mots sont-elles nombreuses?	**Pas de césure**	**Non**
Existe-t-il une adaptabilité dynamique des formats d'affichage?	**Non**	**Non**

2. CHARGE DE TRAVAIL

2.1. BRIEVETE

2.1.1. Actions minimales

(Limiter autant que possible les étapes par lesquelles doivent passer les informaticiens)

	PACBASE	NSDK
Nombre d'étapes dans la sélection des commandes ?	**3 étapes** *Afficher d'abord l'écran de commande, puis sélectionner la ligne commande par une croix, enfin spécifier l'action souhaitée par un code déterminé*	**Une étape** *Placer le curseur sur le menu et l'option voulue* *Ouvrir une boite de dialogue en pressant le bouton droit de la souris*
* Touches raccourcies disponibles ?	**Non**	**Oui**
* Touches fonctions disponibles ?	**Oui**	**Oui**
* Icônes disponibles ?	**NP**	**Oui**
* souris disponible ?	**NP**	**Oui**
Drag and Drop disponible ?	**NP**	**Oui**
L'informaticien doit-il entrer des données qui peuvent être déduites par l'ordinateur ?	**Non**	**Oui**
Lors de la saisie de données, existe-t-il des valeurs par défaut ?	**Oui**	**Oui** *Mais mal spécifiées : par exemple, il doit changer le format et le nom des boutons graphiques que NSDK impose par défaut*
Est-il possible d'atteindre une page ou une fenêtre donnée, sans avoir à les parcourir une à une ?	**Oui**	**Oui** *Le boutons droit de la souris pour accéder directement à la fenêtre des événements au lieu de passer par la boite de dialogue de l'objet*
L'informaticien est-il en mesure de réaliser une action lorsqu'un traitement est engagé?	**Non** *Monotâche*	**Non**
Les options des menus sont-elles doublées par des touches contrôle, fonction ou par des icônes?	**Oui** *Uniquement par des touches Fonction*	**Oui**

2.2 DENSITÉ INFORMATIONNELLE

	PACBASE	NSDK
Les informations nécessaires sont-elles affichées ?	**Oui**	**Oui**
L'information nécessite-t-elle une traduction d'unité ou une interprétation de la part de l'informaticien ?	**Oui** *Interprétation d'abréviations techniques et saisie de codes complexes*	**Oui** *Traduction des termes anglais en français (événements)*
L'informaticien doit-il se rappeler des données d'une page d'écran à une autre?	**Oui**	**Non**
La densité informationnelle est-elle constante d'un écran à un autre ?	**Non** *Dépend du type d'écran : par exemple, densité élevée pour l'écran de maquettage, densité peu élevée pour l'écran de programmation*	**Oui**
Densité globale satisfaisante ?	**Non** *Par exemple, dans la grille de maquettage: les colonnes sont trop proches les unes de autres. Difficultés pour identifier les informations*	**Oui** pour les fenêtres **Non** pour les boites de dialogue (trop d'informations)
Densité locale satisfaisante ?	**Oui**	**Oui**

3. CONTROLE EXPLICITE

(Prise en compte par le système des actions explicites de l'informaticien / Contrôle des informaticiens sur le traitement de leurs actions)

3.1 ACTIONS EXPLICITES

	PACBASE	NSDK
Le système requiert-il une action explicite de validation par l'informaticien suite à une entrée de données ou à une commande ?	**Oui** *Une saisie de commande valide toute la saisie de l'écran : Il est alors impossible de revenir en arrière. Sauf à partir de la session "historisée" où l'informaticien peut récupérer certaines versions*	**Non**
Une action peut-elle entraîner un traitement autre que celui défini ?	**Non**	**Non**
La sélection du menu se déroule-t-elle par une validation explicite ?	**Oui** *Il faut valider la commande par la touche "retour- chariot"*	**Non**
Les entrées de commandes importantes (supprimer, enregistrer...) se terminent-elles par une indication de fin ? (Validation, OK...)	**Non**	**Oui**
L'informaticien peut-il interrompre et reprendre sa tâche sans manipulation complexe, en étant sûr de la retrouver dans l'état où il l'a laissé ?	**Oui** *Pacbase offre une très grande sécurité à ce niveau*	**Oui**

L'informaticien peut-il se retrouver dans l'évolution de son activité: a-t-il les moyens de revenir en arrière, de repartir au début ?	**Oui** *Il peut revenir en arrière tant qu'il n'a pas valider l'écran ou le programme **sur lequel il se trouve***	**Oui**

4.2 PRISE EN COMPTE DE L'EXPERIENCE DE L'INDIVIDU

(Moyens mis en oeuvre pour respecter le niveau d'expérience de l'informaticien)

	PACBASE	NSDK
Existe-t-il des "*raccourcis claviers*", des "*commandes rapides*" ou/et des "*touches fonctions*" ?	**Oui** *Uniquement pour les Touches Fonction*	**Oui**
Existe-t-il des choix d'entrées pas-à-pas ou multiples selon l'expérience des informaticiens ?	**Non**	**Oui** *Par exemple ; on peut choisir dans des options prédéfinies ou les programmer*
Les informaticiens expérimentés ont-ils la possibilité de contourner les techniques de codage ?	**Non**	**Oui**
Ont-ils les moyens de demander un niveau de détail des messages d'erreurs qui soit fonction de leur niveau de connaissance ?	**Non**	**Non**
Les informaticiens expérimentés ont-ils la possibilité de saisir plusieurs commandes avant confirmation ?	**Oui**	**Non**

5. GESTION DES ERREURS

5.1 PROTECTION CONTRE LES ERREURS

(Ensemble des moyens pour détecter et prévenir les erreurs d'entrées de données ou de commandes ou encore les actions aux conséquences néfastes)

	PACBASE	NSDK
Les erreurs sont-elles détectées plutôt lors de la saisie, plutôt lors de la validation? (perturbe la planification)	**Lors de la validation**	**Lors de la validation**
Après chaque session et s'il existe un risque de perte de données, un message le signalant et demandant confirmation de fin de session apparaît-il ?	**Non**	**Oui**
Les labels des champs sont-ils protégés ?	**Oui**	**Oui**
Les aires d'affichage sont-ils protégés ?	**Oui**	**Oui**
Toutes les actions sur l'interface ont-elles été envisagées, et plus particulièrement, les appuis accidentels des touches du clavier ?	**Non**	**Non**

5.2 QUALITE DES MESSAGES D'ERREURS

	PACBASE	NSDK
Les entrées accidentelles de commandes ou de touches fonctions non valides impliquent-elles un affichage d'un message?	**Oui**	**Oui**
Ce message indique-t-il les fonctions et/ou commandes appropriées à cette étape du travail ?	**Non**	**Non**
Les messages d'erreurs sont-ils orientés tâche ?	**Non**	**Non**
Les termes employés dans les messages d'erreurs sont-ils spécifiques / impersonnels / neutres / non réprobateurs ?	**Oui**	**Oui**

5.3 CORRECTION DES ERREURS

	PACBASE	NSDK
L'informaticien dispose-t-il des moyens pour inverser l'action d'une mauvaise commande ?	**Oui**	**Oui**
Suite à une erreur de saisie de commande ou de données, a-t-il la possibilité de corriger la portion de données ou la commande erronée ?	**Oui** *Il peut revenir sur la ligne où se situe l'erreur par exemple*	**Oui**
Au moment de la détection d'erreur de saisie, l'informaticien peut-il apporter les corrections souhaitées?	**Oui**	**Oui**

6. HOMOGENEITE / COHERENCE

	PACBASE	NSDK
La localisation des titres des fenêtres est-elle similaire	**Oui**	**Oui**
Les formats d'écrans sont-ils similaires ?	**Oui**	**Oui**
Les procédures d'accès aux options des menus sont-elles similaires ?	**Oui**	**Oui**
Les messages de guidage, d'erreur et de validation sont-ils construits à chaque fois de manière identique	**Oui**	**Oui**
L'affichage des "prompts" pour la saisie des données et des commandes se fait-il toujours à la même position ?	**Oui**	**Oui**
Le format des champs d'entrée de données des boîtes de dialogue est-il toujours le même ?	**Oui** *D'abord en minuscule, puis, après la validation, transformation en majuscule*	**Oui**

7. SIGNIFIANCE DES CODES ET DENOMINATION

	PACBASE	NSDK
Les titres véhiculent-ils bien ce qu'ils sont sensés représentés ?	**Oui**	**Oui**
* Sont-ils distincts les uns des autres?	**Oui**	**Oui**
Idem pour icônes	**NP**	**Non** *Iconographie peu explicite : on ne saisit pas d'emblée leur signification*
Idem pour Abréviations	**Non**	**Oui**
Idem pour menus et options	**Non**	**Oui**
Idem pour touches contrôles	**NP**	**Non** *Ne correspond pas aux conventions Windows Par exemple : "shift + f4" ouvre un fichier ; alors que c'est CRT + O" sur Windows*
le comportement associé a l'icône produit-il le même effet que le déclen-chement de la commande qui lui est liée?	**NP**	**Oui**
Les codes et abréviations sont-ils significatifs et familiers ?	**Non** *Codes Abscons et arbitraires : après une période d'interruption, l'informaticien oublient ces commandes*	**Non** *Les noms des événements ne sont pas significatifs : difficultés de déterminer leurs actions*
Existe-t-il des commandes à effets multiples ?	**Non**	**Non**
Existe-t-il des paires d'action congruentes ?	**Non**	**Oui**

8. COMPATIBILITE

	PACBASE	NSDK
Les procédures de dialogue sont-elles compatibles avec l'ordre tel que se l'imagine l'informaticien ou celui dont il a l'habitude ?	**Non** *L'informaticien doit s'adapter à la logique prescrite par les canevas de Pacbase*	**Oui** *Interface "à l'écoute" des actions de l'utilisateur : ne prescrit rien*
Les termes employés sont ils familiers aux informaticiens et relatifs à la tâche à réaliser	**Oui** *En fait, le vocabulaire qu'ils emploient dans leur activité est celui de Pacbase*	**Oui** *Correspond au langage employé : Les informaticiens utilisent les codes dans leur tâche*
L'affichage de texte à l'écran est-il conforme aux conventions utilisées pour la présentation de texte sur papier ?	**Oui**	**Non**
Le séquencement des écrans reproduit-il bien la planification de l'action de l'individu ?	**Non** ***C'est la logique procédurale du système qui impose la logique d'action de l'informaticien***	**Oui** *l'informaticien détermine lui-même l'affichage des fenêtres selon l'état de son action*
L'individu a-t-il la possibilité d'interrompre à tout moment sa tâche sans se soucier de l'état dans lequel il l'a laissé et le retrouver dans le même état ?	**Oui**	**Oui**
Le système garde-t-il la trace des actions accomplies par l'informaticien	**Oui** *Journal de toutes les actions*	**Non** *(pas d'historique)*

ANNEXE V

STRUCTURE LOGIQUE D'UN SQUELETTE DE PROGRAMMATION TP

01 Initialisations

Réception

05 - Réception

0510 Réception de l'écran

0512 Traitement appel de doc

0520 Détermination code opération

BOUCLE PAR CATEGORIE

10 Positionnement catégorie

1010 Positionnement catégorie

15 détermination Code Mouvement

20 Contrôle des rubriques

25 Lecture

30 Transfert des rubriques (alimentation/mise à jour)

35 Ecriture pour mise à jour

FIN DE LA BOUCLE (Fin de réception)

40 Gestion de la conversation

4010 alimentataion d'affichage

4020 suite d'écran

4030 abandon de conversation

4040 appel d'un autre écran

End of reception

Affichage

50 Affichage (Initialisation de zones)

5010 Initialisations

BOUCLE PAR CATEGORIE

55 Affichage

5510 Gestion de la catégorie

60 Accès au segment (lecture)

65 Alimentation des Rubriques (alimentation écran)

FIN DE BOUCLE d'affichage

70 Traitements des erreurs (gestions)

8O Affichage

Fonctions appelées

80 Accès physique aux segments (lecture)

81 fonctions appelées par "perform"

ANNEXE VI

JUSTIFICATIONS DE LA RÉFÉRENCE AU MODÈLE GOMS

En quelques mots, rappelons ses principes : GOMS (Card, Moran et Newell, 1983) est un modèle de la structure cognitive que possède l'utilisateur d'un dispositif. Cette structure cognitive est décrite à plusieurs niveaux d'analyse. Elle est composée de quatre unités :

a) **G**oal (but) : représente l'ensemble des buts que l'opérateur veut atteindre (structure symbolique) ;

b) **O**perator (opérateur) : désigne l'ensemble des actions élémentaires (perceptifs, moteurs ou cognitifs) dont l'exécution est nécessaire pour la production d'un changement dans l'état mental de l'individu ou pour effectuer un changement dans l'environnement de la tâche ;

c) **M**ethods (méthodes) : consistent en l'ensemble des moyens procéduraux pour atteindre et réaliser les buts ;

d) **S**election rules (règle de sélection) : correspond à des structures de contrôle de type conditionnel (Si→alors) ou heuristique, permettant de sélectionner la ou les méthodes les plus adéquates pour atteindre le but.

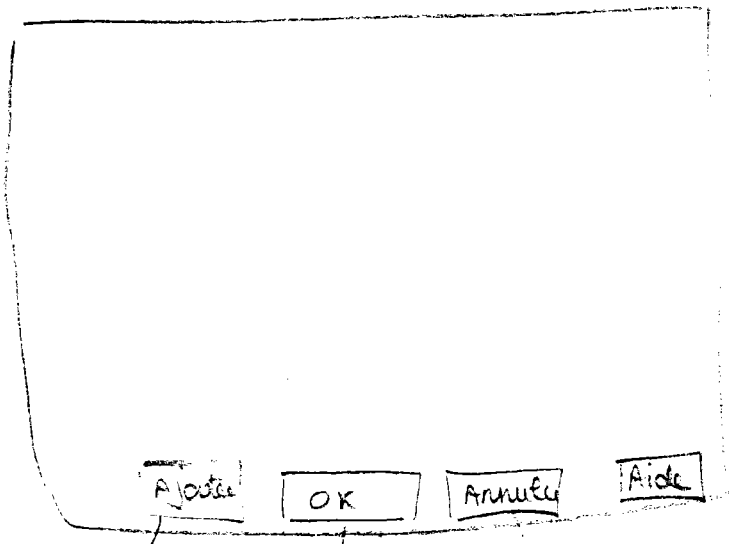
Le choix de ce modèle de représentation comme base de réflexion à notre propre formalisme s'explique pour plusieurs raisons : d'abord, il fournit une description relativement exhaustive des représentations mentales d'un opérateur engagé dans une tâche. Ensuite, il permet d'obtenir une décomposition d'une tâche en une hiérarchie de buts et de sous-buts. Ce qui se révèle utile pour comparer l'organisation de l'activité entre les différents échantillons d'informaticiens. Enfin, c'est un modèle centré sur l'exécution et sur la performance de l'utilisateur.

ANNEXE VII

Exemple de représentation papier de la maquette à réaliser par les novices

le bouton Ajouter

Création d'1 mot



Ajouter

création du mot + ~~initialise~~ efface les champs enlève le relief / mets la couleur noir

création du mot + ferme la fenêtre

ferme la fenêtre

dépense
recette
Montant
commentaire

fenetre D030

tg00 d030

tg00 d030. scr

librairies

tg0000 30. mce

ANNEXE VIII

LEXIQUE TECHNIQUE ET FONCTIONNEL DU MAQUETTAGE

Algorithme
Un algorithme peut être conçu comme un ensemble de commandes permettant la manipulation de formule logique. On peut comprendre son fonctionnement sans savoir comment la description s'écrira en langage machine, comment ce programme sera converti en une séquence d'instructions par la machine abstraite.

Bibliothèque (sur Pacbase)
Elle correspond à une sorte de grosse base de données où sont répertoriées toutes les entités informatiques réalisées dans Pacbase. L'informaticien peut à tout moment venir puiser dans cette réserve pour récupérer des écrans, des programmes,etc.

Charte graphique
Ensemble de normes graphiques indiquant aux informaticiens la manière de composer les fenêtres (par exemple, nombre de pixels entre chaque bouton graphique, mettre la première lettre du libellé d'un bouton en majuscule et la soulignée, ordre de positionnement des boutons...).

Client serveur
Cette architecture informatique met en présence des postes client et serveur, sur lesquels sont répartis les données, les traitements et les interfaces. Le poste client peut effectuer des opérations dans son propre environnement ou bien demander des services au poste serveur.

Contrôle (sur Pacbase)
Un contrôle correspond à un objet graphique (bouton, icône, menu, liste déroulante...) que l'informaticien utilise pour concevoir les fenêtres.

Environnement de conception informatique
Il comprend un ensemble d'outils d'aide à l'exécution, à la documentation, à la lecture, à l'annotation, à la vérification, à l'observation et à la maintenance des programmes. Il a pour but de fournir des aides tout au long du processus de développement d'un projet (de l'analyse des demandes, en passant par les spécifications, à l'implémentation et à la maintenance)

Evénement
Un événement représente les effets spécifiques que génère la sélection d'une commande sur un ou plusieurs composants de l'interface. Ils sont soit proposés par défaut par NSDK (événements Terminate quand une fenêtre se ferme, Selected quand un objet est sélectionné, Lose-focus quand une fenêtre est désactivée au profit d'une autre, Get-focus quand une fenêtre est activée au détriment d'une autre...), soit développés par l'informaticien. Il en existe près d'une soixantaine.

Fenêtre modale (sur NSDK)
L'utilisateur ne peut aller nul part tant qu'il n'a pas validé une commande de cette fenêtre. Il reste bloqué dans ce contexte particulier.

Fenêtre non-modale (sur NSDK)
L'utilisateur à la possibilité d'afficher simultanément plusieurs fenêtres et de passer de l'une à l'autre par une simple clique de souris.

Langages procéduraux
Les langages procéduraux (cobol, fortran et basic) ont été très familiés entre 1960 et 1980. Cobol reste aujourd'hui le langage le plus utilisé pour l'informatique de gestion et de production. Ces langages furent conçus pour décrire pas-à-pas les procédures d'un raisonnement : un petit nombre de verbes (écrire, lire, calculer, aller, achever, etc.) devait suffire constituer le programme d'une application. Ils se prêtent bien à la programmation des algorithmes, en particulier numériques. Un langage procédural se fonde enfin sur la distinction fondamentale des procédures ou traitements et des données qui seront soumises à ces traitements.

Langages déclaratifs
Les langages déclaratifs ignorent la distinction formelle entre données et traitements. Ces deux entités sont des objets déclarés sous forme de listes. L'écriture d'un programme dans un tel langage prend alors la forme d'une succession (non ordonnée a priori) de déclarations énonçant les objets à considérer. Les langages LISP (List Processing) sont parmi les langages déclaratifs les plus connus.

Langages orientés objets
Le paradigme de programmation OO (C++, Smalltalk) est basé sur les notions de classes, d'encapsulation, d'héritage et de transmission de messages. A l'opposé de la programmation classique qui sépare données et procédures, l'approche OO supprime cette dichotomie en intégrant données et fonctions de traitements (appelées méthodes) dans des entités appelées classes. Un objet sera une instance de cette classe et les méthodes seront utilisées pour manipuler ces objets. Par propriété d'héritage, toute classe hérite de la structure et des méthodes de la classe mère. Une des caractéristiques essentielles de ces langages est la réutilisabilité des classes d'objets et des méthodes sur différents projets.

Grille de maquettage (Pacbase)
La grille de maquettage est l'écran de composition des maquettes de Pacbase : elle se présente sous la forme de deux tableaux dans lesquels l'informaticien saisit des codes pour appeler, positionner et paramétrer les rubriques qui composeront les écrans de la maquette.

Librairies (NSDK)
Ce sont des bases spécialisées par fonctions de programmation réutilisables. Comme la bibliothèque de Pacbase, elles mettent à la disposition de l'informaticien tout un ensemble de composants qu'il peut récupérer et insérer à ses projets. Par exemple, la librairie "NS-Date" propose tous les traitements liés aux dates ; "NS-GRAPH", la programmation de graphique (histogrammes...) ; NS-MATH, les fonctions mathématiques (√, log...), etc.

Maquette
Une maquette, contrairement au véritable prototype, est une ébauche plus ou moins complète du produit final. Elle sert à présenter l'interface de dialogue au client. En pratique, dans le domaine des IHM, il est d'usage d'appeler ainsi un ensemble d'objets graphiques organisés pour donner une image fidèle de l'écran tel qu'il sera visible par l'utilisateur du futur logiciel. Les fonctionnalités de la future application ne sont pas implémentées, mais les effets qu'auront leur manipulation sur les autres objets de l'interface sont simulés.

Paradigme événementiel (NSDK)
Il fait référence à l'affichage asynchrone des fenêtres, c'est-à-dire que l'informaticien doit programmer maquette pour que les fenêtres s'ouvrent en fonction des actions et des sollicitations de l'utilisateur.

Paradigme transactionnel ou (conversationnel) (Pacbase)
Par transaction ou conversation, il faut entendre "un tour de parole" régi par la machine. La conversation se tient dans un langage de commandes, efficace et concis, mais abscons.

Procédure stockée
Une procédure stockée est une procédure d'accès automatisée aux bases de données.

Prototype
Un prototype est une version du logiciel réalisant l'ensemble des fonctionnalités de la version définitive, mais réalisé en utilisant des outils (langages de programmation en particulier) permettant une programmation rapide, quitte à sacrifier quelques unes des caractéristiques exigées du produit opérationnel : portabilité, robustesse, voire rapidité.

Pseudo-code
Cela correspond à un mélange de codes informatiques et de commentaires en langage naturel rédigés sur un document de conception.

Rubrique (Pacbase)
C'est la donnée élémentaire d'un fichier de base de données qui est affichée sur un écran. Elle peut être consultée, renseignée ou modifiée par l'utilisateur. Elle constitue l'entité fondamentale du maquettage sur Pacbase.

Site-central
Architecture informatique composée d'un ordinateur central (mainframe) auquel est relié plusieurs terminaux passifs. Ces derniers jouent en quelque sorte le rôle d'interface entre le mainframe et l'individu. Les calculs, les extractions de bases de données, les traitements sont réalisées sur l'ordinateur central par l'intermédiaire de requêtes et de transactions.

Squelettes de programmation
Canevas de fonctions de programmation dans lequel l'informaticien s'insère pour programmer l'application. On distingue le squelette des traitements instantanés (TP) de celui des traitements différés (Batch)