Les approches de conduite de projets informatiques : classification, intégration et recommandations

Mohamed El Louadi, Université du Québec à Trois-Rivières

Introduction

Depuis l'installation des premiers systèmes informatiques de gestion, il y a plus de quatre décennies, plusieurs approches de conduite de projets de développement de systèmes d'information ont émergé et continuent d'émerger. Ces approches vont des plus formelles, telles que les méthodes dites structurées, aux moins formelles quant à leur démarche, telles le prototypage. À cause de cette multitude de méthodes et d'approches, un des problèmes les plus aigus que rencontrent les chefs de projets informatiques est celui du choix de l'approche à suivre.

Pour cette raison, plusieurs chercheurs et praticiens se sont penchés sur le problème du choix en avançant des recommandations destinées à éclairer les chefs de projets sur l'approche la plus adéquate pour un projet donné compte tenu des contingences telles la complexité de l'application (Burns et Dennis, 1985 ; El Louadi *et al.*, 1991) ou le degré de l'incertitude perçue dans le projet (Davis, 1982 ; Naumann et Davis, 1978 ; Naumann *et al.*, 1980).

Dans le modèle des contingences proposé par Burns et Dennis (1985) et El Louadi *et al.* (1991), il est suggéré qu'à un projet impliquant une application à haut niveau de complexité devrait correspondre une méthode structurée conforme aux principes du cycle de vie de Boehm (1981).

Dans le modèle de Davis et ses collègues (Davis, 1982 ; Naumann et Davis, 1978 ; Naumann *et al.*, 1980), à un projet à haut niveau d'incertitude devrait correspondre une démarche par prototypes. À des niveaux intermédiaires de complexité et d'incertitude devraient correspondre des démarches dites « mixtes » (Burns et Dennis, 1985 ; Dennis *et al.*, 1987) qui combinent l'esprit du cycle de vie et celui du prototypage pour définir une approche hybride. El Louadi *et al.* (1991) proposent par ailleurs une correspondance triunivoque entre les types d'application, les types d'approches et des degrés combinés de complexité et d'incertitude.

Si les chercheurs s'entendent sur le fait qu'à un projet assorti d'un niveau d'incertitude élevé devrait correspondre une démarche par prototypage (Davis, 1982), il n'en de-

meure pas moins que l'idée du prototypage reste encore mal définie. Un certain manque de consensus est également présent dans une large mesure dans les méthodes dites structurées. Divers textes pédagogiques et manuels de cours décrivent des démarches de développement en termes de phases, d'étapes ou de niveaux (Boehm, 1981, Castellani, 1985, IGL, 1989 ; Jackson, 1986 ; Whitten *et al.*, 1989 ; Yourdon, 1979). Ces méthodes sont souvent associées au concept générique du modèle du cycle de vie ou « modèle en cascade » (Boehm, 1981).

Dans cet article, nous proposons une classification des approches de conduite de projets informatiques qui couvre les méthodes structurées et les différents types d'approches mixtes qui combinent les démarches par prototypage et l'esprit du cycle de vie. L'intégration des différentes approches à laquelle nous sommes arrivés suite à une revue poussée de la documentation sur le prototypage nous a permis d'arriver à un modèle de sélection de l'approche appropriée dans un contexte de projet donné. Le contexte du projet est défini en fonction de deux contingences : l'incertitude dans le projet telle que définie par Davis et ses collègues et la complexité de l'application à automatiser telle que définie par Burns et Dennis (1985). Ainsi, à des degrés variés d'incertitude et de complexité correspond une approche qui favorise plus ou moins les interactions entre l'utilisateur et l'analyste et la fréquence des itérations à l'intérieur ou entre les phases du cycle de vie.

Si le terme « méthodologie » est souvent utilisé pour se référer à toute méthode, approche, démarche ou stratégie de développement, nous avons choisi, arbitrairement nous le concédons, d'y substituer le terme « approche ». Ce terme sera employé en lieu et place de ce qui est souvent considéré une méthodologie par Burns et Dennis (1985) et Dennis *et al.* (1987), par exemple, qui l'utilisent à la fois pour les méthodes structurées *et* pour le prototypage, par Jenkins (1985) qui l'utilise pour le prototypage seulement et Rochfeld et Moréjon (1986) qui l'utilisent pour Merise, etc. [1] Les méthodes structurées et la démarche par prototypage sont différentes sur plus d'un point. Si l'une est souvent considérée comme une méthode (Whitten *et al.*, 1989), l'autre est toujours vue comme une démarche, une technique, voire même un simple outil de développement (Carey et Mason, 1986). Nous utilisons le terme « méthode » pour toutes les méthodes structurées qui n'incluent pas explicitement le prototypage dans leur définition et l'expression « approche mixte » pour toutes les approches qui ne sont pas exclusivement structurées ou du type prototypage. Comme certaines méthodes peuvent ne pas être assez complètes pour mériter cette appellation, telles HIPO, les DFD, etc., nous les considérons davantage comme des outils, au même titre que les langages de programmation.

De plus nous distinguons entre projet, application et système. Un projet peut englober plusieurs applications. Ces applications peuvent être manuelles ou automatisables. Le terme « système » est surtout appliqué dans le contexte des applications qui sont à automatiser. Nous voudrions également éviter que la notion de structure qui est perçue dans un problème, ou une application, soit confondue avec celle caractérisant les méthodes dites « structurées ». Si la notion de structure dans un problème que nous utilisons rejoint celle définie par Simon (1960), celle relative aux méthodes concerne davantage l'existence préalable d'étapes pré-définies de résolution de problèmes et à laquelle nous reviendrons.

Finalement, nous utilisons le terme « analyste » pour inclure toute personne, homme ou femme, qui est en charge d'une ou de plusieurs activités dans le projet de développement.

Cet article est composé de cinq sections. Dans la première, nous définissons le cycle de vie, qui est aussi appelé « modèle en cascade », en nous inspirant de Boehm (1981). Dans les deuxième et troisième parties, nous définissons, principalement par des exemples les méthodes dites structurées et la démarche par prototypage. Les approches « mixtes », ou « hybrides », font l'objet de la discussion de la typologie de la quatrième partie qui est sui-

vie par des recommandations concernant le choix de l'approche à suivre basées sur un modèle qui incorpore des facteurs relatifs à la complexité de l'application et à l'incertitude perçue au début du projet.

I. Le cycle de vie ou modèle en cascade

Le cycle de vie du développement d'un projet informatique a été présenté par Royce en 1970 et vulgarisé par Boehm (1981). Voir la figure 1. Alors que la version de Boehm (1981) est souvent utilisée comme la référence (Rochfeld et Moréjon, 1989), d'autres versions comportent un ordonnancement différent des tâches et des activités au niveau de chaque phase (McKeen, 1983). Déjà en 1970, Murdick avait dénombré au moins dix-sept versions différentes officiellement publiées et documentées.

Dans le cycle de vie, l'on ne passe d'une phase à une autre que si la première a été achevée, validée et documentée. Le passage au fil des phases est souvent linéaire et ne comporte que quelques itérations entre certaines phases, étapes ou niveaux. Ces itérations sont pré-programmées dans le cycle dont la structure se veut en parallèle avec le degré de structure perçu dans le problème (Colter, 1982). Dans les méthodes qui se réclament du cycle de vie, l'analyste a très peu de marge de manœuvre s'il choisit de se conformer strictement à la démarche. Mais tel n'est pas le cas en réalité. Ainsi les méthodes de Jackson (1986) ou de Yourdon et Constantine (1979), par exemple, sont rarement appliquées à la lettre (Kozar, 1989 ; McKeen, 1983). Plusieurs organismes ou firmes privées développent leurs propres standards, méthodes et recommandations (Naumann *et al.*, 1980). Ces méthodes sont souvent inspirées d'autres démarches dont les éléments ou les outils sont greffés et/ou adaptés. Dans la méthode Merise par exemple (Rochfeld et Moréjon, 1989 ; Tardieu *et al.*, 1983), le succès du processus de développement dépend directement de la fidélité d'application de ces phases (Matheron, 1987). Il est à noter, cependant, que rien dans le cycle de vie ne proscrit le retour en arrière. En fait, la représentation graphique de la démarche que Boehm avait donnée rend explicite l'existence d'itérations entre les phases. Il n'en demeure pas moins que ces itérations sont restreintes au niveau de deux phases contiguës. Ainsi une itération est possible entre la phase de conception détaillée et celle de la conception globale, entre la phase de codage et celle de la conception détaillée, mais pas entre la phase de conception globale et celle du codage. Les conséquences d'une telle « myopie [2] » dans le développement ont été amplement discutées par Henson et Hughes (1991).

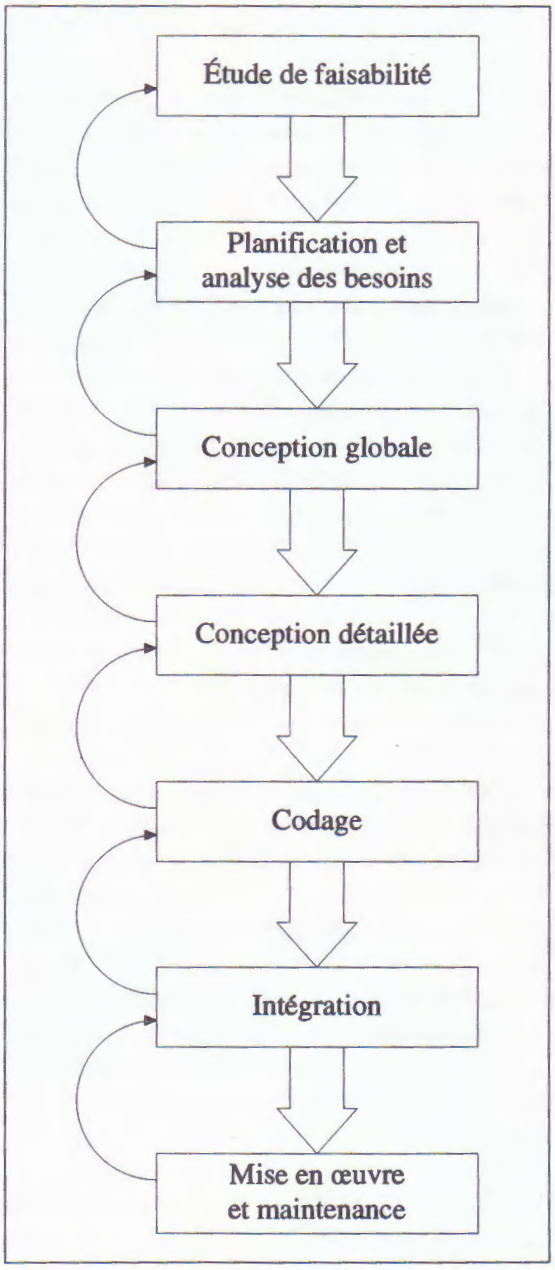


Figure 1 : *Le cycle de vie d'un projet de développement d'un système d'information (D'après Boehm, 1981).*

Dans l'approche du cycle de vie, un problème est d'abord défini et l'étendue de ce problème détermine à son tour le projet de l'équipe informatique. Ce problème est alors décomposé en sous-problèmes. La décomposition se base souvent sur la nature des fonctions que les applications composant le projet recouvrent, d'où le terme « découpage fonctionnel » dans la méthode Castellani par exemple. Le découpage peut se faire en plusieurs étapes successives selon le niveau de complexité du problème et est d'autant plus facilité que le problème lui-même le permet. Plus le projet se prête à un tel découpage et plus il est dit structuré. Le concept de structure dans un projet est relatif et déteint souvent sur la ou les applications qu'il engendre.

En résumé, si un projet est assez complexe pour justifier un découpage fonctionnel et assez structuré pour rendre une telle démarche aisée, alors le cycle de vie est appliqué.

Le cycle de vie n'est pas une technique en soi. Il n'est pas une méthode non plus, quoique certains auteurs (Burns et Dennis, 1985 ; Dennis *et al.*, 1987 ; Whitten *et al.*, 1989) le qualifient souvent comme tel ou le confondent carrément avec les méthodes structurées (Plyler et Kim, 1993). Le cycle de vie est un esprit, une stratégie de gestion de projets, au mieux une philosophie. Des méthodes sont greffées au cycle de vie et sont utilisées au niveau d'une ou de plusieurs phases. Ces méthodes à leur tour se basent sur des outils et des techniques. C'est le nombre de combinaisons méthode-techniques-outils possibles qui ont mené aux constatations de Murdick (1970), plus tard reprises par Whitten *et al.* (1989), qu'il existe « plusieurs » cycles de vie. Nous n'allons pas nous aventurer à identifier ces différentes combinaisons. Nous allons par contre définir les méthodes dites structurées dans les paragraphes qui suivent.

II. Les méthodes structurées

La documentation sur la gestion des projets informatiques s'accorde unanimement sur le fait que les méthodes structurées sont toutes basées sur le principe de la décomposition hiérarchique. Cela avait en fait commencé au niveau le plus décisif du cycle de vie : la phase de codage ou de la programmation (Goldberg, 1986). On commença d'abord par parler de programmation structurée puis de programmation modulaire dont le principe fondamental est de décomposer un programme (ou module) en sous-programmes (ou sous-modules) selon des critères et des règles précises (abolition des GOTO, une seule fonction par module, etc.) (Dijkstra, 1968 ; Wirth, 1971). Puis vinrent les méthodes qui s'adaptent à des phases antérieures à celles de la programmation : les phases d'analyse (conception globale dans la figure 1) et de conception. Selon Colter (1982), ces méthodes sont définies en termes non pas de phases mais de concepts et d'outils. Les concepts essentiels sont qu'un problème doit être décrit selon une approche descendante, allant du global au détail, de l'abstrait au concret. Les outils sont les diagrammes de flux de données (DFD), les dictionnaires de données, les diagrammes de description des traitements, etc.

Les méthodes structurées d'analyse les plus connues sont celles de DeMarco (1978), Gane et Sarson (1979) et Yourdon (1979). Ces méthodes se basent sur des outils de description des traitements qui, si elles utilisent des formalismes différents, s'apparentent toutes aux DFD.

Les méthodes structurées de conception les plus connues sont celles de Yourdon et Constantine (1979), Page-Jones (1980), Warnier-Orr (Orr, 1977 ; Warnier, 1973) et Jackson (1986). Ces méthodes utilisent également des outils de description graphique similaires aux DFD.

Toutes ces méthodes sont orientées sur l'étude des traitements. D'autres méthodes structurées sont davantage préoccupées par l'étude des données. Celles-ci incluent la méthode de Martin et Finkelstein (*Information Engineering*, 1981), de Flavin (*Information Modeling*, 1981), le développement

orienté-objet (Shlaer et Mellor, 1988) et le modèle entité-relations (Chen, 1976).

D'autres méthodes structurées plus complètes, dont Merise (Rochfeld et Morejon, 1989 ; SADT (Ross, 1977), SSADM (SSADM, 1990) ou ASDM (voir Jaakola et Drake, 1991) modélisent à la fois les données et les traitements tout en assurant l'indépendance qui doit prévaloir entre les deux (ANSI, 1975). Parmi les méthodes les moins complètes et qu'on serait tenté de classifier en tant qu'outils, notons la méthode HIPO (*Hierarchical Input Process Output*) d'IBM (1974) et la Méthode de Conception des Programmes (MCP) de Dominique Warnier (1973).

Dans toutes ces méthodes, l'usage d'outils graphiques de description et d'analyse, tels les DFD, est surtout accompli dans un souci de description et d'analyse. Si le recours à ces outils a souvent été justifié par le besoin d'un outil de communication analyste-utilisateur comme c'est le cas de SADT — voir le titre de l'article de Ross (1977) par exemple —, cette communication reste limitée à l'utilisateur et à l'analyste. Tous deux doivent s'imaginer le futur système et tous deux forment des attentes en fonction de leur vision de ce système. L'expérience et la documentation sur les systèmes d'information en témoigne, a montré que ces visions correspondent rarement l'une à l'autre et presque jamais à la réalité que sera le système une fois celui-ci implanté. L'expérience a de plus montré que ces outils n'étaient pas toujours compréhensibles pour les utilisateurs (Andrews, 1983 ; Berrisford et Wetherbe, 1979 ; Earl, 1978). Cet état de fait était surtout évident dans le contexte de projets qui ne se prêtaient pas à une décomposition hiérarchique qu'elle soit axée sur les traitements ou sur les données. De tels projets sont souvent complexes, très peu structurés et font partie d'un environnement sans cesse changeant. La rigidité des méthodes structurées, souvent greffées dans le cycle de vie d'un projet ajoutait au problème. L'alternative se présenta sous la forme d'une démarche que l'on qualifia de prototypage.

III. La démarche par prototypage

L'application de la démarche par prototypage au développement des systèmes informatiques fut inspirée des architectes et des spécialistes en génie électrique. La démarche originale ne contient aucune rigueur dans le sens linéaire et séquentiel du cycle de vie. Si dans le cycle de vie les phases sont distinctes, dans la démarche par prototypage, la notion de phases n'existe pas. En fait, les phases de Boehm sont confondues en une étape répétée plusieurs fois (Parker, 1983). Cette notion d'itérations qui n'était présente qu'entre deux phases successives dans le modèle de Boehm devient centrale dans la démarche par prototypes et peut être appliquée à n'importe quel niveau du développement. Au début de son adaptation aux projets informatiques, la démarche par prototypes était souvent comparée à une « révolution » ou plus encore, à un « nouveau paradigme » (Jenkins, 1985 ; Naumann et Jenkins, 1982). La nouvelle approche créa un engouement tel que Galletta et Naumann (1982) dégagèrent pas moins de 43 articles publiés entre 1973 et 1981 dans leur brève revue annotée de la littérature concernant le prototypage. La recherche que nous avons effectuée dans la banque de données ABI/Inform a abouti à 1728 titres dont 42 pour la seule année de 1994.

Le principe fondamental de la démarche veut que l'analyste et l'utilisateur interagissent continuellement alors même que le système est en train d'être développé de sorte que l'utilisateur puisse réagir à un problème dès que celui-ci prend naissance (Naumann et Jenkins, 1982). Ainsi les interactions incluent souvent non seulement l'analyste et l'utilisateur, mais le système qui devient de plus en plus concret. L'utilisateur et l'analyste n'imaginent plus, ils réagissent à un programme réel, non pas un programme dans le sens de Brooks (1975) mais une maquette de programme. Les interactions s'arrêtent lorsque l'utilisateur est complètement satisfait.

L'analyste qui opte pour le développement par prototypage développe d'abord un prototype, puis analyse, conçoit, implante et

réimplante, teste et retourne au point de départ de l'analyse en fonction des vœux et souhaits de l'utilisateur. Dès le début, l'utilisateur peut émettre des commentaires basés sur un système vraisemblablement opérationnel qui n'est en fait qu'une maquette. L'utilisateur utilise cette maquette pour dégager les besoins de l'utilisateur et ajuster le tir au fil des itérations. Plus l'analyste interagit avec l'utilisateur et plus il a des chances de pouvoir répondre à ses attentes plus rapidement et plus efficacement (Berrisford et Wetherbe, 1979 ; Naumann et Jenkins, 1982). Contrairement aux méthodes qui se réclament du cycle de vie, la fréquence des interactions dans le prototypage est la règle plutôt que l'exception.

Si plusieurs reproches ont été faits à l'encontre des méthodes structurées, la démarche par prototypage n'est pas dépourvue de failles non plus. Un des reproches qu'on lui fait dans la littérature a un rapport direct avec sa définition : il n'y a pas de prescriptions d'utilisation (Carey et Mason, 1983 ; Naumann et Jenkins, 1982). Si les méthodes structurées étaient souvent jugées trop rigides, la démarche par prototypes est tombée dans l'autre extrême. Carey et Mason (1983) offrirent une revue de la littérature sans toutefois proposer une définition consensuelle quant à la nature propre de ce que la démarche par prototypes est réellement. Carey et Mason ont néanmoins le mérite d'avoir offert une description de la démarche en tant que technique ou outil.

Il ne fut pas longtemps avant que les chercheurs et académiciens se rendirent compte que la nouvelle approche ne pouvait être le substitut de son antithèse représentée par les méthodes structurées. Le besoin d'une synthèse, ou d'une combinaison, a d'abord été ressenti par Bally *et al.* en 1977. La synthèse a dégagé les points forts de chaque approche pour trouver l'alliage « idéal » dépendant des caractéristiques du projet et des préférences des analystes concepteurs. Notre revue de la littérature sur le prototypage a révélé que la démarche peut être utilisée de plusieurs manières. Elle peut également être incorporée dans plusieurs phases du cycle de vie (Bally *et al.*, 1977 ; Carey et Mason, 1983 ; Davis *et al.*, 1988 ; Johnson, 1983). C'est ainsi qu'est apparue une nouvelle génération d'approches dites approches « mixtes » ou « hybrides » (Dennis *et al.*, 1987).

IV. Les approches mixtes

Notre revue de la littérature sur les projets de développement d'applications informatiques nous a amené à la conclusion que plusieurs auteurs se réfèrent à une même méthode sous différentes appellations. Un exemple peut être trouvé dans les travaux de chercheurs tels que Davis (1982), Naumann et Davis (1978), Naumann *et al.* (1980). Dans ces travaux, les chercheurs ont développé un modèle qui, compte tenu du niveau d'incertitude présent dans un projet, indique la meilleure approche à suivre pour amorcer un projet d'informatisation. La constance des éléments du modèle d'un article à l'autre nous a forcé à croire que les approches dont les auteurs parlent ne peuvent qu'être les mêmes. Pourtant les approches sont nommées différemment d'un article à l'autre. Ainsi, par exemple, ce qui est appelé « découverte expérimentale des besoins de l'utilisateur » dans l'un (Davis, 1982) est nommé « prototypage » dans l'autre (Naumann et Davis, 1978), les deux termes étant également ambigus.

Un autre exemple est révélé par les différents termes utilisés pour qualifier une approche qui n'est somme toute rien d'autre que le prototypage ou une variante du prototypage. Berrisford et Wetherbe (1979) et Wetherbe (1982) utilisent le terme *développement heuristique* (*heuristic development*). McCracken et Jackson (1982), Floyd (1983) et Davis *et al.*, 1988) utilisent le terme *développement évolutionniste* (*evolutionary development*). Ness (1975), Nolan (1974) et Hurst *et al.* (1982) parlent de l'*approche intermédiaire* (*the middle-out approach*) et Keen (1980) et Keen et Gambino (1983) utilisent la phrase *approche du design adaptatif* (*the Adaptive Design approach*). Elam (1980) pré-

fère le terme *processus de développement itératif* (*iterative design process*) et Burns et Dennis (1985) parlent de *conception en phases* (*phased design*). Les descriptions de toutes ces méthodes telles qu'offertes par ces auteurs ne font que rappeler l'absence de méthode a priori, c'est-à-dire la présence du prototypage.

Il semble qu'à un détail près, toutes ces variétés du prototypage ont un dénominateur commun : celui des itérations. La différence entre les différents types de prototypage est le niveau auquel ils sont appliqués en référence au cycle de vie ; certains s'appliquent à une seule phase, d'autres à plusieurs phases et d'autres encore couvrent toutes les phases du cycle de vie.

Il est à rappeler qu'une approche mixte est un mélange de modèle structuré et de prototypage. Le principe de « mélange » a été d'abord éclairci par Johnson (1983) qui a essayé d'apporter une réponse à la question concernant les aspects d'un système devraient être assujettis à un développement par prototypage. Il a conclu que, dans certaines situations, le cycle de vie pourrait être rendu plus approprié, voire plus efficient, s'il était augmenté d'une certaine dose de prototypage. Il a présenté une classification à quatre niveaux pour catégoriser les différents usages qui sont faits de cette nouvelle approche (une synthèse). Trois de ces niveaux ont été adoptés par Cerveny *et al.* (1985) et incorporés dans un modèle de développement des systèmes par El Louadi *et al.* (1991). Deux de ces niveaux correspondent à l'approche heuristique de Berrisford et Wetherbe (1979) et à l'approche adaptative de Keen (1980).

Dans la même lancée que Carey et Mason (1983), Johnson (1983) et Cerveny *et al.* (1985), nous avons entrepris de répertorier les différentes versions de la démarche par prototypage que nous avons trouvées dans la documentation depuis l'article de Royce (1970) jusqu'à celui de Plyler et Kim (1993) —un total de 72 livres et articles les plus cités. Nous aidant de la catégorisation de Johnson (1983), nous sommes parvenu à faire ressortir cinq familles d'approches mixtes. Nous avons par ailleurs noté que la vaste majorité des articles sur le prototypage ne manquaient pas de le définir par rapport au cycle de vie ou à une ou plusieurs de ses phases. À l'exception de la version de Martin (1982) où le prototypage est total puisqu'il est appliqué par l'utilisateur lui-même, il nous a paru que le prototypage, dans la plupart de ses versions, servait soit de base à une application d'un modèle en phases proche du cycle de vie ou de complément à un cycle de vie en bonne et due forme. Les cinq familles d'approches mixtes qualifiées en termes de types de prototypage sont esquissées dans la figure 2. C'est sur la base de cette figure que les différentes versions du prototypage sont regroupées et définies dans l'ordre croissant de leur inclusion dans le cycle de vie.

IV.1. Prototypage de type 1

Dans ce type de prototypage, la démarche prototypage n'est utilisée que pour aider l'analyste à concevoir l'interface du futur système. Il s'agit de concevoir des maquettes d'écran, des menus et des rapports imprimés. Le prototypage de type 1 est équivalent au prototypage de niveau 1 de Johnson (1983) et de Cerveny *et al.* (1985). Il est similaire mais pas identique au développement heuristique de Berrisford et Wetherbe (1979) en ce sens que l'interface ne communique avec aucun aspect interne du système. Si l'utilisateur a l'impression que des données sont entrées, affichées ou mises à jour, ces données font partie d'une simulation dont le seul objectif est d'étudier l'interface humain-machine et sa dynamique afin que l'analyste puisse prendre compte du comportement de l'utilisateur face à la machine. A-t-il tendance à préférer F1 ou F3 pour l'aide ? A-t-il tendance à avoir davantage recours aux touches de direction qu'à la souris ? etc.

Comme il est montré dans la figure 2, le prototypage de type 1 (étapes 11-14) intervient assez tard dans le cycle de vie. En fait, il a lieu dans la phase de la conception détaillée après la décomposition fonctionnelle

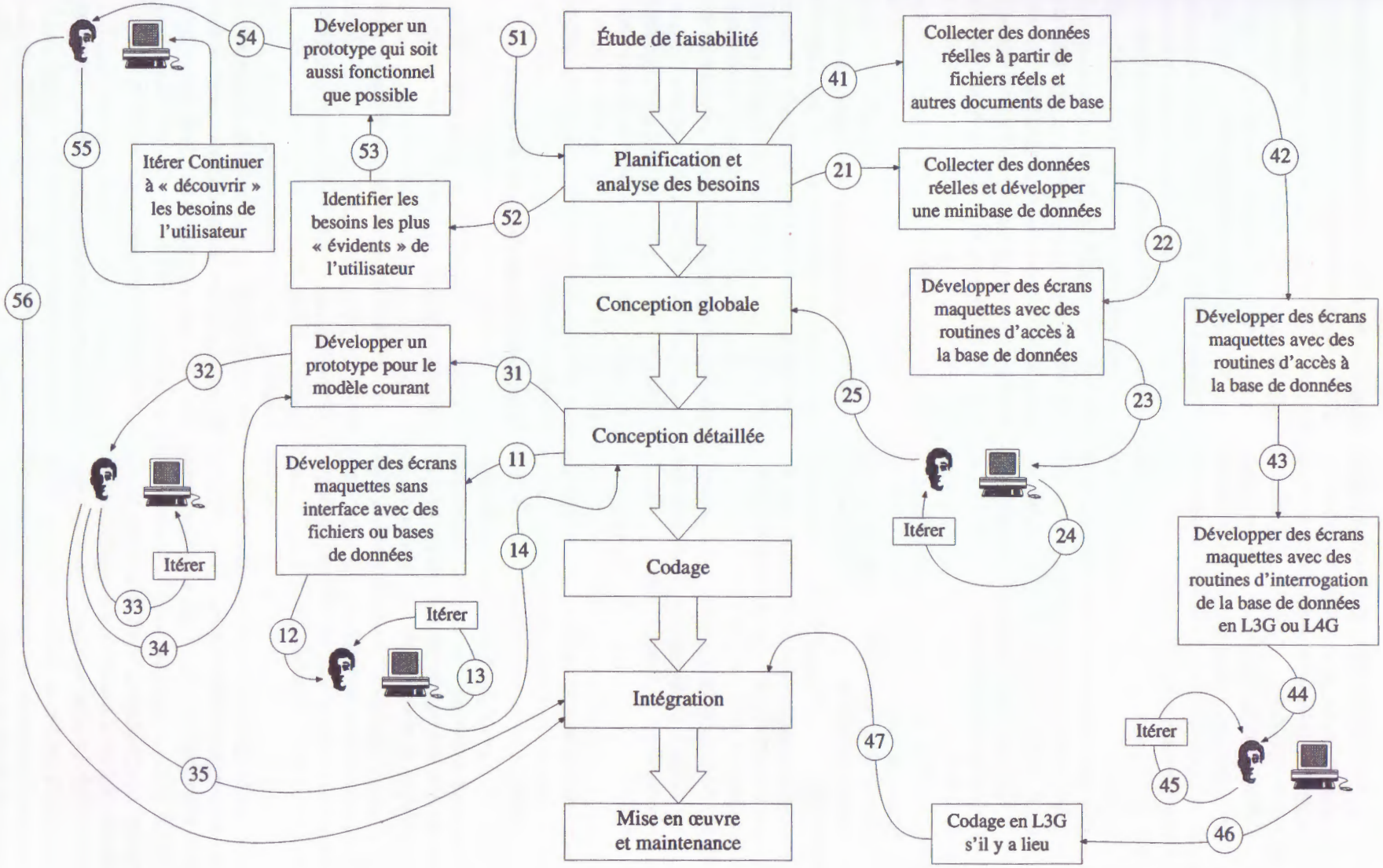


Figure 2 : *Les cinq différentes familles d'approches relativement au cycle de vie de Boehm.*

de la phase de conception globale. Les itérations se font au fur et à mesure que l'utilisateur suggère des modifications ou des améliorations et continuent jusqu'à ce que celui-ci soit complètement satisfait de l'aspect extérieur du système. Comme il est montré dans la figure 2, c'est le seul endroit où les interactions ont lieu par rapport au cadre général du cycle de vie.

Un outil souvent utile dans ce type de prototypage est le modèle de conception des dialogues et des écrans de Foley (voir Myers, 1980). Ce modèle décompose la structure des interfaces en composantes spécifiques : le niveau conceptuel, le niveau sémantique et le niveau lexical.

Le système qui a servi de prototype ne sera jamais utilisé ni incorporé dans la version finale ; il n'aura servi que comme outil de communication analyste-utilisateur en remplacement des dessins d'écran que l'on trouve dans la méthode Castellani par exemple. En cela, il inclut également les prototypes « jetables » de Gomaa et Scott qui font aussi partie des prototypes de type 2.

IV.2. Prototypage de type 2

Cette famille de démarches par prototypage correspond au niveau 2 de Johnson (1983) et au niveau 1 de Cerveny *et al.* (1985). On notera que tout comme dans le prototypage de niveau 1 de Cerveny *et al.*, le développement heuristique chevauche nos deux types de prototypage. Dans ces cas, notre distinction a surtout été guidée par le positionnement de chaque type par rapport au cycle de vie. Cette famille inclut aussi les prototypes « jetables » (*throwaway*) popularisés par Gomaa et Scott (1981) et le prototypage par simulation de Iivari et Karjalainen (1989). Elle correspond au développement heuristique de Berrisford et Wetherbe (1979) dans le cas où l'interface est testée sur des données réelles emmagasinées dans le système sous forme de fichiers test ou minibases de données. La conception de ces fichiers et de cette base de données fait elle-même partie de l'objectif de ce type de prototypage qui ne se réduit pas à concevoir l'interface comme c'est le cas dans le prototypage de type 1.

Ce type du prototypage (étapes 21-25) est surtout utilisé pour traduire les besoins de l'utilisateur en spécifications du système, vérifier la faisabilité de ces spécifications, tester les différentes conceptions offertes et éventuellement choisir la « meilleure » conception. Il se positionne donc au plus tard au niveau de la conception. Son utilisation est justifiée par les résultats d'études qui ont montré que de 60% à 80% des erreurs de développement ont lieu au niveau de cette phase (Boar, 1984).

S'il se préoccupe essentiellement des besoins en informations de l'utilisateur, ce type de prototypage ne néglige pas l'aspect performance de l'interface (temps de réponse, usage et accès aux mémoires auxiliaires de l'ordinateur, étude des accès aux fichiers pour des fins d'optimisation après normalisation, etc.). Si dans la version de Gomaa et Scott (1981), les prototypes peuvent être remplacés par un système réel (d'où la notion de prototype « jetable »), Davis *et al.* (1988) avancent que les prototypes utilisés peuvent être directement greffés dans la version finale du système puisqu'ils ont déjà été validés.

IV.3. Prototypage de type 3

C'est l'approche correspondant au design en phases de Dennis *et al.* (1985). Elle correspond également au niveau 2 de Cerveny *et al.* (1985) et au niveau 3 de Johnson (1983). C'est l'approche mixte par excellence où une série de développements par prototypes s'effectue au niveau des modules (ou unités organiques) tout en gardant l'esprit de séquence et de linéarité du cycle de vie au niveau du développement de tout le système.

Dans le prototypage de type 3 (étapes 31-35), l'analyste entame la conception en suivant l'esprit du cycle de vie. L'étude des besoins et la conception se font selon le prin-

cipe du découpage fonctionnel en utilisant des méthodes structurées adaptées à cette phase : DFD, entrevues structurées, etc. Chacune des décompositions qui en résulte fait l'objet d'une étude détaillée basée sur le principe du prototypage. Elle est d'abord programmée en utilisant des langages de programmation très évolués tels que les langages de quatrième génération (L4G) et est présentée à l'utilisateur selon un calendrier logique. D'abord ce sont les modules d'entrée des données qui sont développés. Ensuite viennent les applications de validation et de mise à jour. Enfin viennent les applications de restitution (impression, affichage, etc.). Ainsi, l'analyste choisit d'implanter les parties du système qui lui sont les plus compréhensibles.

Cette approche est surtout recommandée lorsque l'application peut être décomposée en des sous-applications relativement indépendantes (Dennis *et al.*, 1987). Un exemple d'une telle approche est donné par Shoval (1988) et sa méthode ADISSA (Architectural Design of Information Systems Based on Structural Analysis).

IV.4. Prototypage de type 4

C'est la démarche correspondant à l'approche incrémentale de Hirsch (1985) et à une moindre mesure à l'approche intermédiaire de Ness (1975). Dans cette approche, nous connaissons la plupart des besoins de l'utilisateur, mais bien moins que dans le prototypage de niveau 3 (design en phases). Le développement incrémental se résume à utiliser une série de prototypes du système définitif et à ajouter à leur fonctionnalité et à leur performance par incréments. Une métaphore qui s'impose serait l'application de plusieurs couches de peinture à un pan de mur par exemple.

Les prototypes de niveau 4 (étapes 41-47) diffèrent de ceux du niveau 2 sur deux plans. Ces prototypes sont moins que complets en termes de fonctionnalité mais représentent plus que de simples simulations. D'abord, une modélisation réelle des données est effectuée selon une approche similaire à celle du modèle conceptuel des données de Merise ou L.A.P.A.G.E. Des entités ou groupements primaires de données sont formés afin de concevoir une première version des modèles de données externes. Des routines d'interrogation des données sont également programmées à l'aide d'un langage de troisième génération (L3G tel que COBOL), d'un langage de quatrième génération (L4G tels que Focus, PacBase, Quickbuild, etc.) ou d'un langage d'accès à une base de données du type SQL par exemple. C'est le volume des transactions, des données et la performance attendue du système qui détermine le bien- fondé de l'utilisation d'un langage de troisième génération plutôt que de quatrième génération. Dans le cas où un L4G est utilisé, cette démarche s'apparenterait à celle de Gomaa et Scott (1981) et la série de prototypes se réduirait à une succession de prototypes « jetables » (Davis *et al.*, 1988).

IV.5. Prototypage de type 5

Ce type correspond au niveau 4 de Johnson (1983), au niveau 3 de Cerveny *et al.* (1985), au design adaptatif ou version 0 de Keen (1980), à l'approche intermédiaire de Ness (1975) et Hurst *et al.* (1982). C'est aussi l'équivalent du design évolutionniste présenté brièvement à la fin de l'article de McCrackem et Jackson (1982) et plus amplement expliqué par Floyd (1983). C'est le type de prototypage le plus complet à l'exception de celui de Martin (1982) où c'est l'utilisateur qui développe le système et qui n'est pas représenté dans la figure 2.

Dans ces approches (étapes 51-56), l'analyse de faisabilité est totalement éliminée. L'analyste passe directement à la phase de l'analyse et détermination des besoins. À partir des besoins les plus « évidents » ou les mieux connus, il implante des prototypes de travail (à ne pas confondre avec les maquettes ou les prototypes « jetables »). Ces prototypes constituent la base même à laquelle l'analyste s'accroche pour converger, d'itération en ité-

ration, vers les besoins « réels » de l'utilisateur. Il n'y a aucune distinction de phases, l'application la mieux servie par ce genre de prototypage peut être complexe mais n'est certainement pas structurée. Par voie de conséquence, la décomposition fonctionnelle est rarement possible. Chaque prototype (itération) est utilisé pour essayer différentes configurations mais aussi pour découvrir, voire modifier les besoins des utilisateurs (Whitten *et al.*, 1989).

En résumé, l'analyste développe une implantation partielle du système qui répond à des besoins connus et le prototype est utilisé pour découvrir les autres besoins. Le fait que l'analyste sache rarement à l'avance à quoi le futur système va ressembler a valu à cette approche d'être comparée à un projet de recherche et développement où on peut s'attendre à tout y compris à l'échec. Un exemple réel d'une approche de prototypage de niveau 5 est donné par la méthode Multiview de Avison et Wood-Harper (1986).

V. Quelle approche utiliser ?

Le choix de l'approche à utiliser pour la conduite d'un projet de développement d'une application informatique dépend de facteurs tant managériaux, organisationnels que techniques. Souvent ce choix n'est pas manifeste et certaines organisations n'ont même pas le choix parce qu'elles ont déjà institutionnalisé une approche standard ou parce qu'elles se font développer leurs systèmes par des cabinets d'études externes. D'autres facteurs peuvent s'ajouter qui sont relatifs à la tâche à laquelle le projet se rattache, aux compétences humaines et techniques engagées ainsi qu'aux ressources disponibles.

Nous avons pu déterminer que certains projets nécessitaient une décomposition fonctionnelle dû à leur complexité. Nous avons également pu déterminer que dans certaines situations, une telle décomposition pouvait ne pas être possible. D'un autre coté, force nous est de constater que le cycle de vie dans sa rigueur et sa rigidité ne facilitait pas la conduite de projets qui sont peu ou pas structurés. Dans ce cas, le prototypage est souvent utilisé parce qu'il favorise les interactions utilisateur-analyste-système tout en multipliant les itérations au niveau d'une ou de plusieurs activités. Ces activités sont si clairement délimitées dans le cycle de vie qu'elles se regroupent en phases homogènes. La différence entre le cycle de vie et le prototypage, rappelons-le, se situe aux deux niveaux de la fréquence des interactions utilisateur-analyste-système d'une part et du nombre d'itérations intra- ou inter-phases d'autre part. La fréquence des interactions aide à diminuer l'incertitude dans le développement. Cette incertitude est due entre autres choses à la faible structure perçue dans l'application à automatiser. Ainsi, en même temps que les interactions augmentent, les itérations deviennent plus nombreuses. On se démarque par conséquent de plus en plus de l'esprit du cycle de vie classique pour nous orienter vers les différents types de prototypage ou approches « mixtes ». Ces différents types de prototypage, s'ils contiennent tous une certaine dose d'itérations et d'interactions, ils les utilisent à différents niveaux du développement. Ainsi, plus on s'éloigne du type 1 et on se rapproche du type 5 et plus le nombre d'activités touchées par ces itérations augmente. Le nombre des itérations et des interactions augmente en fonction de plusieurs critères qui ont souvent été regroupés en deux facteurs essentiels : l'incertitude perçue dans le projet et la complexité de l'application (Alter et Ginzberg, 1978 ; Burns et Dennis, 1985 ; Curtis *et al.*, 1988 ; Davis, 1982 ; Dennis *et al.*, 1987 ; McFarlan, 1981 ; Naumann et Davis, 1978 ; Naumann *et al.* 1980) tels que définis au tableau I.

V.1. L'incertitude perçue dans le projet

La définition selon laquelle l'incertitude est conçue comme le résultat d'un manque d'information (Galbraith, 1977) a été reprise et incorporée dans le contexte de projets informatiques par Naumann et Davis (1978). Parmi les sources d'incertitude identifiées par les chercheurs en gestion de projets informa-

tiques figure le degré de structure de la tâche à informatiser (Davis, 1982), le degré de compétence des utilisateurs qui sont la source principale de l'information dont l'analyste a besoin (Alter et Ginzberg, 1978 ; Curtis *et al.*, 1988 ; Davis, 1982) et la compétence des analystes et leur habileté à comprendre l'information qui leur est fournie par l'utilisateur (Davis, 1982 ; Larsen et Naumann, 1992 ; McFarlan, 1981). De plus, certaines applications et projets sont notoirement entachés d'un degré élevé d'incertitude, comme par exemple un système pour lequel il n'existe aucun référentiel préalable (Burns et Dennis, 1985 ; El Louadi, 1991). Ces facteurs sont identifiés au tableau I.

En accord avec la théorie de Galbraith, Davis (1982), Davis et Naumann (1978), El Louadi *et al.* (1991) avancent que l'adoption d'une démarche de développement qui favorise l'interaction entre l'analyste (demandeur d'informations) et l'utilisateur (pourvoyeur d'informations) pour pallier l'incertitude est recommandée. Ainsi, la fréquence des interactions analyste-utilisateur est d'autant plus encouragée que l'incertitude augmente. Étant donné que les approches mixtes sont celles qui favorisent ces interactions à divers degrés et à diverses étapes du cycle de développement comme nous l'avons discuté plus haut, il n'est pas surprenant que l'usage de ces approches soit associé à des degrés croissants d'incertitude tel que cela est illustré à la figure 3.

V.2. La complexité de l'application

La complexité, dans son sens le plus large, a été définie par Simon (1969) et Weinberg (1975) comme étant une fonction du nombre d'objets composant un système et de la difficulté de décrire d'une manière statique et/ou dynamique les interrelations entre ces objets. Le courant cartésien veut par ailleurs que la meilleure stratégie de résolution d'un problème complexe soit la décomposition récursive de ce problème en sous-problèmes avec l'hypothèse que les sous-problèmes sont moins complexes du fait même de la décomposition.

Ainsi, les facteurs qui définissent la complexité d'un projet doivent obligatoirement inclure une notion de « complication » (Landry, 1988). Ils incluent la taille du projet (Burns et Dennis, 1985 ; El Louadi, 1991), le nombre des utilisateurs pour lesquels diverses vues du même futur système peuvent exister, le volume des informations à générer par le futur système et le degré de « décomposabilité » de ce système. Par ailleurs, et en parallèle avec la notion de l'incertitude, certains systèmes sont naturellement complexes.

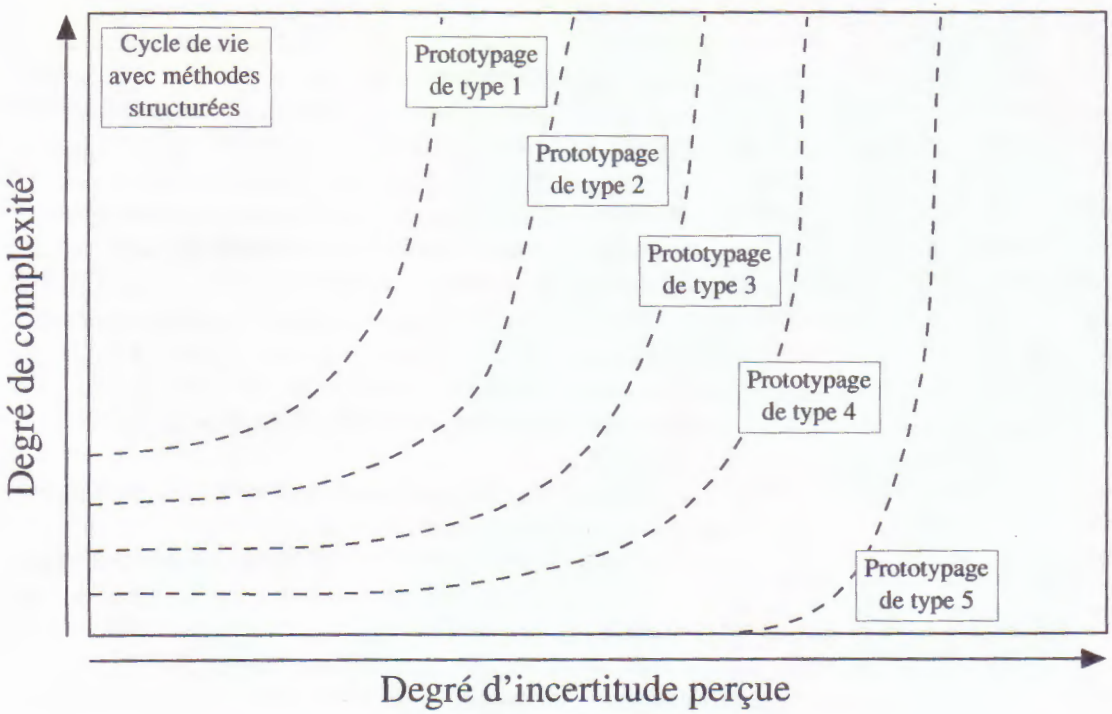
Conscients du fait qu'aucun mécanisme déterministe basé sur des critères de décomposition clairs n'était inclus dans la démarche pro-cartésienne de Simon et Weinberg, les chercheurs en gestion de projets informatiques (Boehm, 1981 ; Ross, 1977 ; etc.) ont accueilli cette démarche et l'ont appliquée aux systèmes complexes tout comme cela avait été fait dans d'autres domaines tels le génie électrique.

À des degrés croissants de complexité, l'on souhaiterait être à même de disposer d'approches riches en outils de décomposition (DFD, HIPO, etc.) qui nous permettraient de voir le détail sans perdre de vue le global (les liens entre les différents niveaux des DFD). Ce raisonnement découle de la théorie de contrôle des systèmes (control theory, Greensite, 1970) qui veut justement que pour garder le contrôle sur un système complexe, l'on doit disposer d'un système doté d'une structure au moins égale au degré de complexité du système à contrôler. Ainsi, dans la figure 3, plus le degré de complexité augmente et plus on tend vers des approches structurées ayant la rigueur normative du cycle de vie.

Conclusion

Le lecteur aura remarqué que plusieurs des approches chevauchent une ou plusieurs familles et que les lignes démarcatives de la figure 3 sont en pointillés. Seule l'approche de Hirsch s'est avérée être assez unique pour justifier la création d'un type de prototypage

Figure 3 :
Le modèle du choix de l'approche en fonction des deux contingences définies dans le tableau I



Légende

Méthodes « structurées »	SADT, Merise, ASDM, SSADM, etc.	
Prototypage de type 1	— niveau 1 de Johnson	— développement heuristique de Berrisford et Wetherbe
Prototypage de type 2	— niveau 2 de Johnson — simulation de Iivari et Karjalainen	— niveau 1 de Cerveny *et al.* — « jetables » de Gomaa et Scott
Prototypage de type 3	— niveau 3 de Johnson — niveau 2 de Cerveny *et al.* — design en phases — ADISSA de Shoval	
Prototypage de type 4	— incrémentale de Hirsch	
Prototypage de type 5	— Multiview — adaptatif de Keen — niveau 4 de Johnson — niveau 3 de Cerveny et al. — évolutionniste — « jetables » de Gomaa et Scott	— intermédiaire de Ness

différent. En effet, à la différence des approches du type 3, elle va plus loin que la phase de conception. À la différence des approches du type 5, elle passe d'abord par l'étude de faisabilité. Au-delà des approches de type 5 se trouve le prototypage dans sa forme la plus pure, celle dans laquelle les interactions analyste-utilisateur-système se font sans l'ana-

Figure 2 :
Les facteurs dont il faudra tenir compte lors du choix de l'approche à utiliser

Incertitude

Degré de structure :
- est-ce qu'il existe des règles de gestion claires correspondant à chaque situation qui pourrait se présenter dans la plupart des tâches à automatiser ?
- existe-t-il une séquence bien définie dans le déroulement des opérations et des tâches à automatiser ?

Utilisateurs :
- connaissent-ils bien les tâches à automatiser ?
- ont-ils une bonne connaissance de leurs besoins en informations ?
- peuvent-ils communiquer clairement leur savoir et besoins ?
- sont-ils prêts à participer au développement ?

Analystes :
- ont-ils de l'expérience dans des applications similaires ?
- ont-ils de l'expérience dans le développement des systèmes en général ?
- peuvent-ils communiquer avec les utilisateurs en un langage simple et clair ?
- ont-ils de l'expérience avec plusieurs approches de développement ?
- en combien de langages peuvent-ils programmer ?

Type d'application :
- est-ce une application de comptabilité, de marketing, de prévision budgétaire, etc ?
- est-ce une application par lots, en temps réel, en transactionnel, en conversationnel ?

Type de projets :
- est-ce une amélioration d'un système existant, l'adaptation d'un autre logiciel ou progiciel, un système personnalisé, un système à développer entièrement ?

Complexité

Taille du projet :
- quel est le nombre de fonctions que le futur système est supposé supporter ?
- est-ce que le système doit être opérationnel sur micro-ordinateur ou sur mini-ordinateur ?

Utilisateurs :
- est-ce que le système est destiné à être utilisé par un ou plusieurs utilisateurs ?
- est-ce que le système est destiné à plusieurs utilisateurs dans la même unité ?
- est-ce que le système est destiné à être utilisé par un cadre supérieur ou un cadre moyen ?
- est-ce que les utilisateurs sont sophistiqués ou profanes en ce qui concerne l'informatique ?

Information générée :
- quel est le volume de l'information que le système va générer ?
- quel est le degré de complexité de l'information que le système va générer ?
- à quelle fréquence cette information doit-elle être rendue disponible ?

Type d'application :
- est-ce que l'application est décomposable ?
- est-ce que les différentes décompositions peuvent être gérées comme des applications indépendantes ?
- est-ce que l'application est à être utilisée dans plusieurs endroits géographiques différents ?
- quel est l'étendue des tâches à informatiser par rapport à celles qui vont rester manuelles ?
- quel est le degré d'inter-dépendance entre les tâches automatisables et les tâches manuelles ?

Type de système :
- est-ce un système de traitement des transactions, un système interactif d'aide à la décision, un système de gestion de l'information, un système expert, etc. ?

lyste puisque l'utilisateur devient le concepteur. Si le cycle de vie est partout omniprésent dans les approches mixtes, ce n'est que dans le prototypage au sens de Martin (1982) qu'on en fait abstraction totale. Nous n'avons pas inclus cette version du prototypage total parce qu'elle peut difficilement s'insérer dans un contexte de conduite de projets. De plus, il est vrai que le prototypage n'est pas totalement exclu des méthodes structurées. Plusieurs auteurs reconnaissent l'utilité de l'utilisation « occasionnelle » du prototypage dans Merise (voir Rochfeld et Moréjon, 1989, pages 26-38) ou Whitten *et al.* (1989).

Un des problèmes que nous avons détectés dans la documentation sur les facteurs de l'incertitude et de la complexité est le manque de discernement entre ce qui définit l'un qui est exclusif de l'autre et vice-versa. Les limitations de notre catégorisation du tableau I est le reflet du faible consensus qui prévaut quant à la validité des construits théoriques en question. Notre contribution a été d'aller au-delà des construits théoriques en proposant aux gestionnaires de projets un cadre de référence, sous la forme de questions, qui nous l'espérons, les aidera à mieux planifier leurs projets.

Dans cet article, nous avons présenté une discussion sur les problèmes associés à la difficulté d'identifier les différentes approches de développement des systèmes d'information. Basé sur notre revue de la littérature, nous avons identifié le cycle de vie comme étant une démarche de conduite d'un projet qui regroupe plusieurs approches dites structurées. Le cycle de vie a été utilisé comme cadre de référence —ou ossature— afin de définir différentes approches de développement de systèmes informatiques. Nous avons également identifié plusieurs types de démarches par prototypage qui viennent se greffer à une ou plusieurs phases du cycle de vie et définissent ainsi l'ensemble des approches « mixtes ». L'esprit de rigueur normative et séquentielle du cycle de vie se perd au fur et à mesure que le prototypage s'y insère jusqu'à le couvrir complètement. Nous avons proposé un modèle qui subdivise cette évolution du cycle de vie aux approches « mixtes » en six types : les méthodes structurées et cinq types de prototypage. Ces approches sont sélectionnées en fonction de facteurs spécifiques à l'entreprise en fonction de la version du cycle de vie et des méthodes standards en vigueur dans leur environnement ou pays : Merise pour la France, SSADM pour le Royaume-Uni, la méthode de la British Computer Society (BCS) pour le Canada, SADT ou la méthode de Jackson (JAD) pour les États-Unis, Vorgehensmodell pour l'Allemagne et Dafne pour l'Italie.

Notes

1 . Il est certes possible que le terme « méthodologie » en français ne soit pas la traduction directe du terme « methodology » en anglais. Rochfeld et Moréjon (1989) utilisent « méthode » et Rochfeld et Tardieu (1983) utilisent « methodology » en se référant à la même « méthode » Merise. Par ailleurs, Michel Galinier n'utilise le terme « méthodologie » (en français) qu'une seule fois dans tout l'ouvrage (IGL, 1989, p.2) en se référant à ce qui semble être ce que nous appelons « approche ».

2. Nous avons trouvé particulièrement révélatrice la citation de Rouxel (Shadok) sous la définition du mot « méthode » dans le livre sur SADT (une méthode structurée rappelons-le) IGL (1989) : « Il est beaucoup plus intéressant de regarder où l'on ne va pas, pour la bonne raison qu'où l'on va, il sera toujours temps d'y regarder quand on y sera. » (p. 2).

Bibliographie

ALTER, S. et GINZBERG, M., « Uncertainty in MIS Implementation », *Sloan Management Review*, 1978, p. 23-31.

ANDREWS, W., « Prototyping Information Systems, *Journal of Systems Management*, 1983, p. 16-18.

ANSI-SPARC DATA BASE TASK GROUP, INTERIM REPORT, *ACM SIGMOD Record*, New York, 1975.

AVISON, D. E. et Wood-Harper A. T., « Multiview—An Exploration in Information Systems Development », *The Australian Computer Journal*, 1986, p. 174-179.

BALLY, L., BRITTAN, J. et WAGNER, K. H., « A Prototype Approach to Information System Design Development », *Information & Management*, 1977, p. 21-26.

BERRISFORD, T. et WETHERBE, J. C., « Heuristic Development : A Redesign of Systems Design », *MIS Quarterly*, 1979, p. 11-19.

BOAR, B. H., *Application Prototyping*, New York, John Wiley, 1984.

BOEHM, B. W., *Software Engineering Economics*, Englewood Cliffs, N.J., Prentice-Hall, 1981.

BROOKS, F. P., *The Mythical Man-Month*, Addison-Wesley, 1975.

BURNS, R. N. et DENNIS, A. R., « Selecting the Appropriate Application Development Methodology », *Data Base*, 1985, p. 19-23.

Carey, T. T. et Mason R. E .A., « Information System Prototyping : Techniques, Tools, and Methodologies », *Infor*, 1983, p. 177-191.

CASTELLANI, X., *Méthode générale d'analyse d'une application informatique*, Paris, Masson, 1985.

CERVENY, R. P., GARRITY, E. J. et SANDERS, G. L., « The Application of Prototyping to Systems Development : A Rationale and Model », *Journal of Management Information Systems*, 1985, p. 52-62.

CHEN, P. P-S, « The Entity-Relationship Model—Toward a Unified View of Data », *ACM Transactions on Database Systems*, 1976, p. 9-36.

CURTIS, B., KRASNER, H. et ISCOE, N., « A Field Study of the Design Process for Large Systems », *Communications of the ACM*, 1988, p. 1268-1287.

DAVIS, G.B., « Strategies for Information Requirements Determination », *IBM Systems Journal*, 1982, p. 4-30.

DAVIS, A. M., BERSOFF, E. H. et COMER, E. R., « A Strategy for Comparing Alternative Software Development Life Cycle Models », *IEEE Transactions on Software Engineering*, 1988, p. 1453-1461.

DEMARCO, T. *Structured Analysis and System Specification*, New York, Yourdon Press, 1978.

DENNIS, A.R., BURNS, R. N. et GALLUPE, R. B., « Phased Design : A Mixed Methodology for Application Systems Development », *Data Base*, 1987, p.31-37.

DIJKSTRA, E.W., « The Structure of "THE" Multiprogramming System », *Communications of the ACM*, 1968, p. 341-346.

EARL, M. J., « Prototype Systems for Accounting, Information, and Control », *Data Base*, 1982, p.39-46.

EL LOUADI M., POLLALIS, Y. A. et TENG, J. T. C., « Selecting a Systems Development Methodology : A Contingency Framework », *Information Resources Management Journal*, 1991, p. 11-19.

ELAM, P. G., « Choosing between System Development Alternatives », *Journal of Systems Management*, 1980, p. 36-40.

FLAVIN, M., *Fundamental Concepts of Information Modeling*, New York, Yourdon Press, 1981.

FLYNN, D. J. et FRAGOSO-DIAZ, O., « Conceptual EuroModelling : How do SSADM and MERISE Compare ? », *European Journal of Information Systems*, 1993, p. 169-183.

FLOYD, C., « A systematic look at prototyping, Approaches to Prototyping », *Proceedings of a Working Conference on Prototyping*, 1983.

GALBRAITH, J., Organization Design, Reading, MA : Addison-Wesley Pub. Co, 1977.

GALLETTA, D. F. et NAUMANN, J. D., « Annotated Bibliography of Prototyping for Information Systems Development », *University of Minnesota Working Paper Series : MISRC-WP-82-12*, 1982, p. 1-34.

GANE, C. et SARSON, T., *Structured Systems Analysis : Tools and Techniques*, Englewood Cliffs, N. J., Prentice-Hall, 1979.

GOLDBERG, R., « Software Engineering : An Emerging Discipline », *IBM Systems Journal*, 1986, p. 334-353.

GOMAA, H. et SCOTT, D., « Prototyping as a Tool in the Specification of User Requirements », *Proceedings of the 5th IEEE*

International Conference on Software Engineering, 1981, p. 333-342.

GREENSITE, A. L., *Control Theory*, New York, Spartan Books, 1970.

HENSON, K. L. et Hughes C.T., « A Two-Dimensional Approach to Systems Development », *Journal of Information Systems Management*, 1991, p. 35-43.

HIRSCH, E., « Evolutionary Acquisition of Command and Control Systems », *Program Manager*, 1985, p. 18-22.

HURST, E. G, NESS, D. N., GAMBINO, T. J. et JOHNSON, T. H., « Growing DSS : A Flexible, Evolutionary Approach », dans *Building Effective Decision Support Systems*, Englewood Cliffs, N. J., Prentice-Hall, 1982.

IBM, « HIPO — A Design Aid and Documentation Technique », White Plains, New York, *IBM Form No. GC20-1851*, 1974.

I.G.L. Technology, S.A.D.T. Un langage pour communiquer, Éditions Eyrolles, Paris, 1989.

IIVARI, J. et KARJALAINEN, M., « Impact of Prototyping on User Information Satisfaction During the IS Specification Phase », *Information & Management*, 1989, p. 31-45.

JAAKOLA, J. E. et Drake K. B., « ASDM : The Universal Systems Development Methodology », *Journal of Systems Management*, 1991, p. 6-11.

JACKSON, I. F., *Corporate Information Management*, nglewood Cliffs, N. J., Prentice-Hall, E1986.

JENKINS, A. M., « Prototyping : A Methodology for the Design and Development and Development of Application Systems », *Spectrum*, 1985, p. 1-8,.

JOHNSON, J. R., « A Prototypical Success Story », *Datamation*, 1983, p. 251-256.

KOZAR, K. A., « Adopting Systems Development Methods : An Exploratory Study », *Journal of MIS*, 1989, p. 74-86.

KEEN, P. G. W., « Adaptive Design for Decision Support Systems », *Data Base*, 1980, p.15-25, 1980.

KEEN, P. G. W. et GAMBINO, T .J., « Building a Decision Support System : The Mythical Man-Month Revisited », dans J. L. Bennett (ed.), *Building Expert Systems*, Reading, Massachusetts, p. 133-172, 1983.

LANDRY, M., « Les problèmes organisationnels complexes et le défi de leur formulation », *Revue canadienne des sciences de l'administration*, 1988, p. 34-48.

LARSEN, T. J. et NAUMANN, J. D., « An Experimental Comparison of Abstract and Concrete Representations in Systems Analysis », *Information & Management*, 1992, p. 29-40.

MARTIN, J., *Application Development without Programmers*, Englewood Cliffs, N. J., Prentice Hall, 1982.

MARTIN, J. et FINKELSTEIN, C., *Information Engineering*, vol. 1 et 2, Savant Institute, 1981.

MATHERON, J-P., *Comprendre MERISE*, Paris, Éditions Eyrolles, 1987.

MCCRACKEN, D. D. et JACKSON, M.A., « Life Cycle Concept Considered Harmful », *ACM SIGSOFT Software Engineering Notes*, p. 29-32, 1982.

MCFARLAN, F. W., « Portfolio Approach to Information Systems », *Harvard Business Review*, 1981, p. 142-150.

MCKEEN, J. D., « Successful Development Strategies for Business Application Systems », *MIS Quarterly*, 1983, p. 47-66.

MURDICK, R. G., « MIS Development Procedures », *Journal of Systems Management*, 1970, p. 22-26.

MYERS, W., « Computer Graphics : The Human Interface », *Computer*, 1980.

NAUMANN, J. D. et DAVIS, G. B., « Contingency Theory Approach to Systems Life Cycle Management », *Proceedings of the Second Software Life Cycle Management Work*, 1978, p. 63-65.

NAUMANN, J. D. et JENKINS, A. M., « Prototyping : The New Paradigm for Systems Development », *MIS Quarterly*, 1982, p. 29-44.

NAUMANN, J. D., DAVIS, G. B. et MCKEEN, J. D., « Determining Information Requirements : A Contingency Method for Selection of a Requirements Assurance Strategy », *The Journal of Systems and Software*, 1980, p. 273-281.

NESS, D. N, « Interactive Systems : Theories and Design », *Joint Wharton/ONR Confe-*

rence—Interactive Information and DSS, Philadelphie, The Wharton School, Department of Decision Sciences, University of Pennsylvania, 1975.

ORR, K. T., *Structured Systems Development*, New York, Yourdon Press, 1977.

PAGE-JONES, M., *The Practical Guide to Structured Systems Design*, New York, Yourdon Press, 1980.

Parker, D. C., « The Evolution of Management Decision Support Systems », *Information Systems*, 1983, p. 56-65.

Plyler R.W. et Kim Y-G., « Methodology Myths, Four Tenets for Systems Developers », *Information Systems Management*, 1993, p. 39-44.

ROCHFELD, A. et MORÉJON, J., *La Méthode Merise : gamme opératoire*, Les Éditions d'organisation, 1989.

ROCHFELD, A. et TARDIEU, H., « Merise : An Information System Design and Development Methodology », *Information & Management*, 1983, p. 143-159.

ROSS, D. T., « Structured Analysis (SA) : A Language for Communicating Ideas », *IEEE Transactions on Software Engineering*, 1077, p. 16-34.

ROYCE, W. W., « Managing the Development of Large Software Systems : Concepts & Techniques », *Proceedings of the Wescon Conference*, 1970.

SHLAER, S. et MELLOR, S. J., *Object-Oriented Systems Analysis : Modeling the World in Data*, Englewood Cliffs, N. J., Prentice-Hall, 1988.

SHOVAL, P., « ADISSA : Architectural Design of Information Systems Based on Structured Analysis », *Information Systems*, 1988.

SIMON, H. E., *The New Science of Management Decisions*, New York, Harper and Row, 1960.

SIMON, H.E., *The Science of the Artificial*, Cambridge, MIT Press, 1969.

SSADM, *Structured Systems Analysis & Design Model, Version 4 Reference Manual*, Oxford, NCC Blackwell, 1990.

TARDIEU, H., ROCHFELD, A. et COLETTI, R., *La Méthode Merise, principes et outils*, Paris, Les Éditions d'organisation, 1983.

WARNIER, J-D., *Les Procédures de traitement et leurs données*, Paris, Les Éditions d'organisation, 1973.

WEINBERG, G. M., *An Introduction to General Systems Thinking*, John Wiley & Sons, NY, 1975.

WETHERBE, J. C., « Systems Development : Heuristic or Prototyping », *Computer-World*, 1982

WHITTEN, J. L., BENTLEY, L. D. et BARLOW, V. M., *Systems Analysis & Design Methods*, Homewood, IL, Irwin, 1989.

WIRTH, N., « Program Development by Stepwise Refinement », *Communications of the ACM*, 1971.

YOURDON, E. N., *Managing the Structured Techniques*, Englewood Cliffs, N. J., Prentice-Hall, 1979.

YOURDON, E. N. et CONSTANTINE, L., *Structured Design : Fundamentals of a Discipline of Computer Program and Systems Design*, Englewood Cliffs, N. J., Prentice-Hall, 1979