ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THÈSE PRÉSENTÉE À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DU
DOCTORAT EN GÉNIE
Ph.D.

PAR
NORMAND SÉGUIN

INVENTAIRE, ANALYSE ET CONSOLIDATION DES PRINCIPES FONDAMENTAUX DU GÉNIE LOGICIEL

MONTRÉAL, LE 17 OCTOBRE 2006

CETTE THÈSE A ÉTÉ ÉVALUÉE
PAR UN JURY COMPOSÉ DE :

M. Alain Abran, directeur de thèse
Département de génie logiciel et des technologies de l'information
à l'École de technologie supérieure

M. Tony Wong, président du jury
Département de génie de la production automatisée
à l'École de technologie supérieure

M. Witold Suryn, professeur
Département de génie logiciel et des technologies de l'information
à l'École de technologie supérieure

M. Robert Dupuis, professeur
Département d'informatique
à l'Université du Québec à Montréal

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC
LE 28 SEPTEMBRE 2006
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

INVENTAIRE, ANALYSE ET CONSOLIDATION DES PRINCIPES FONDAMENTAUX DU GÉNIE LOGICIEL

NORMAND SÉGUIN

SOMMAIRE

Depuis 1970, un certain nombre d'auteurs se sont intéressés à définir le noyau de base du génie logiciel. Ces auteurs s'entendent sur le fait que la discipline doit s'appuyer sur des principes fondamentaux qui seraient moins sensibles à l'évolution rapide de la technologie. Contrairement aux autres disciplines du génie, le génie logiciel ne peut s'appuyer sur les principes de la physique, ainsi, le génie logiciel doit identifier ses propres principes fondamentaux. Plus de 300 principes ont été recensés au niveau des travaux antérieurs sur l'identification des principes du génie logiciel. Depuis plus de 30 ans, les listes de principes s'accumulent sans qu'il y ait un consensus de la part de la communauté sur un ensemble précis de principes. Le nombre élevé de principes publiés bloque l'avancement de la recherche sur le sujet. Cette thèse traite de front 308 principes recensés depuis 1970 à l'aide d'une méthodologie analytique originale afin de filtrer l'ensemble des principes et de ne conserver que ceux qui satisfont à des critères d'identification précis. En premier, un bilan historique des travaux est fait afin de bien cerner la problématique. Par la suite, les objectifs de la recherche sont établis, ainsi que la méthodologie composée de quatre étapes. La première phase consiste à développer le cadre conceptuel nécessaire à la thèse. Par la suite, la deuxième phase applique les critères individuels d'identification aux 308 principes. La troisième phase applique deux critères d'ensemble aux principes retenus à la phase 2, catégorise les principes et effectue des liens avec les processus de la norme ISO/IEC 12207. La quatrième phase procède à une évaluation du degré de couverture des principes retenus à la phase 3 en fonction des éléments du modèle d'ingénierie, dans un premier temps, et en fonction des normes du génie logiciel de l'IEEE. Cette thèse a permis d'obtenir une liste réduite de 34 principes qui répondent aux critères d'identification. De plus, ces 34 principes supportent bien les éléments de base du modèle d'ingénierie, ainsi que les normes du génie logiciel de l'IEEE. Il est maintenant possible de poursuivre la recherche sur les principes du génie logiciel avec la liste réduite obtenue de cette thèse.

INVENTAIRE, ANALYSE ET CONSOLIDATION DES PRINCIPES FONDAMENTAUX DU GÉNIE LOGICIEL

NORMAND SÉGUIN

ABSTRACT

Since the 70's, a number of authors have shown interest in defining the basic core of the software engineering discipline on the tenet that the discipline should be based on fundamental principles that should be more stable than the quickly evolving technologies. However, in contrast to other engineering disciplines, software engineering cannot be based on the principles of physics, thus, the software engineering must identify its own fundamental principles.

More than 300 proposed principles were surveyed in publications dealing with the identification of the principles of software engineering. For over 30 years, the lists of principles have multiplied but without developing a consensus on behalf of the community on a precise set of principles. This high number of proposed principles is an handicap to further research on this topic.

This thesis analyses the 308 principles surveyed since 1970, using an analytical methodology in order to filter this set of the principles and to preserve only those which satisfy criteria fundamental principles. This analytical research methodology consists of four phases. In the first phase, the conceptual framework necessary to the thesis is developed. In the second phase, the individual criteria of identification are applied to the 308 principles. In the third phase, the two overall criteria are applied to the principles retained from phase 2; categorizes the principles; and makes link with standard ISO/IEC 12207. In the fourth phase, an evaluation is done on the degree of coverage of the principles retained from phase 3 with respect to both the elements of the process model of engineering, and to the IEEE software engineering standards

The outcome of this thesis is a reduced list of 34 principles which satisfy the set of criteria identified in our research methodology. Moreover, this thesis illustrates how these 34 principles support well the basic elements of the engineering process model, as well as the IEEE software engineering standards. This reduced list can then be used for further research on software engineering fundamental principles.

REMERCIEMENTS

Je remercie M. Alain Abran, professeur au département de génie logiciel de l'ETS, pour m'avoir invité à poursuivre mes études de doctorat sous sa direction. J'ai pu compter à chaque étape du développement de cette thèse sur sa grande disponibilité, sur ses précieux conseils et sur sa vision globale de la recherche. De plus, M. Abran a su me motiver à l'avancement de cette thèse dont le sujet ne fut pas toujours facile à développer. Je le remercie aussi pour son temps, son dévouement et sa patience tout au long des cinq années de mes études au doctorat.

Je tiens à remercier M. Tony Wong, professeur au département de la production automatisée de l'ETS, d'avoir accepté d'être le président du jury. Également, M. Witold Suryn, professeur au département de génie logiciel et M. Robert Dupuis d'avoir aussi accepté d'être membre du jury.

Je remercie M. Yves Gingras, professeur au département d'histoire de l'UQAM et titulaire de la Chaire de recherche du Canada en histoire et sociologie des sciences, pour son aide précieux afin de bien situer les fondements entre les principes, les concepts et lois dans le contexte scientifique. Je remercie aussi mes amis Gilles Gauthier, Philippe Gabrini et Louis Martin de m'avoir encouragé à entreprendre des études de doctorat et d'avoir cru à ma volonté de les compléter.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS ET SIGLES

IEEE	Institute of Electrical and Electronics Engineers
PMI	Project Management Institute
PMBOK	Project Management Body of Knowledge
SGBD	Système de gestion de bases de données
SWEBOK	Software Engineering Body of Knowledge

INTRODUCTION

Depuis sa première mention à une conférence de l'OTAN en 1968[1], le génie logiciel a connu une multitude de développements et une évolution importante. Les outils de développement, les méthodes et les techniques évoluent sans cesse à l'intérieur de courts cycles de vie entraînant une désuétude rapide des méthodes et des techniques étroitement reliées aux technologies (Bourque et al. 2002). Tandis qu'un bon nombre de contributeurs à la discipline travaillent au développement de nouveaux outils et de nouvelles méthodes, d'autres s'intéressent aux fondements même du génie logiciel à la recherche d'une base plus stable et plus durable sur laquelle la discipline pourrait appuyer son développement. Ghezzi et al. (2003) soulignent que la discipline ne peut pas se baser uniquement sur les méthodes, les techniques et les outils pour établir ses fondements. La discipline doit plutôt s'appuyer sur un ensemble de principes fondamentaux qui seraient plus durables et moins sensibles à la désuétude occasionnée par l'évolution rapide des technologies. Ainsi, Ghezzi et al. (2003) proposent à la figure 1 un modèle des relations entre les principes, les techniques, les méthodologies et les outils.

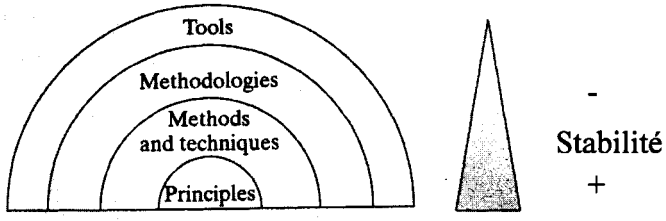

Figure 1 Relations entre les principes et les techniques, méthodologies et outils
Ghezzi et al. (2003)

Les principes représenteraient la base ou le noyau central sur lequel s'appuieraient les techniques, les méthodes et les outils. Également, Ghezzi et al. (2003) soulignent que la

[1] International Conference on Software Engineering at Garmisch, West Germany, 1968

connaissance, par l'ingénieur logiciel, des principes le guiderait tout au long de sa carrière à sélectionner les méthodes, les techniques et les outils appropriés au contexte particulier de chacun des projets. L'identification d'un ensemble de principes qui constitueraient cette base est importante pour fournir les assises nécessaires aux composants de la discipline.

Les principes comme base stable du corpus des normes

Le génie logiciel dispose maintenant d'un corpus de normes et de standards qui systématisent le travail de plusieurs milliers de professionnels du logiciel. James Moore (1998) fait état que le corpus des normes totalisait, en 1997, plus de 300 normes sous la responsabilité de plus de 50 organismes qui ont des visions et des préoccupations différentes envers le génie logiciel. Le corpus des normes s'est constitué petit à petit en observant les meilleures pratiques et en les formalisant. À leur tour, les normes systématisent et formalisent les pratiques des professionnels de l'industrie (figure 2).

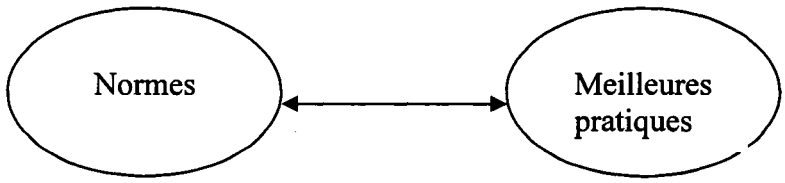

Figure 2 Relations entre les normes et les meilleures pratiques

Cependant, cette relation a des conséquences importantes. Ainsi, Moore (1998) souligne que vu d'une façon globale, le corpus des normes n'est pas bien intégré, manque de cohérence et souffre de certains chevauchements et de contradictions. Comme de plus en plus d'organisations se fient aux normes de l'industrie pour améliorer leurs processus et la qualité de leurs produits logiciels, ce constat préoccupe les organismes responsables du corpus des normes.

Moore (1998) souligne que dans d'autres disciplines du génie, il est possible d'établir des liens précis entre les normes et un ensemble de principes scientifiques et d'ingénierie qui fournissent des balises aux normes. Cependant, en génie logiciel, les organismes sont confrontés à la difficulté de faire ce même type de liens. D'une part, à cause de la nature intangible du produit (le logiciel), non sujet aux lois de la physique, par exemple. D'autre part, la discipline du génie logiciel est relativement jeune et n'a pas encore atteint un niveau de maturité. Pour combler cette lacune, Moore (1998) et Bourque et al. (2002) proposent un modèle, présenté à la figure 3, qui situe le rôle important des principes fondamentaux par rapport aux normes et aux pratiques.

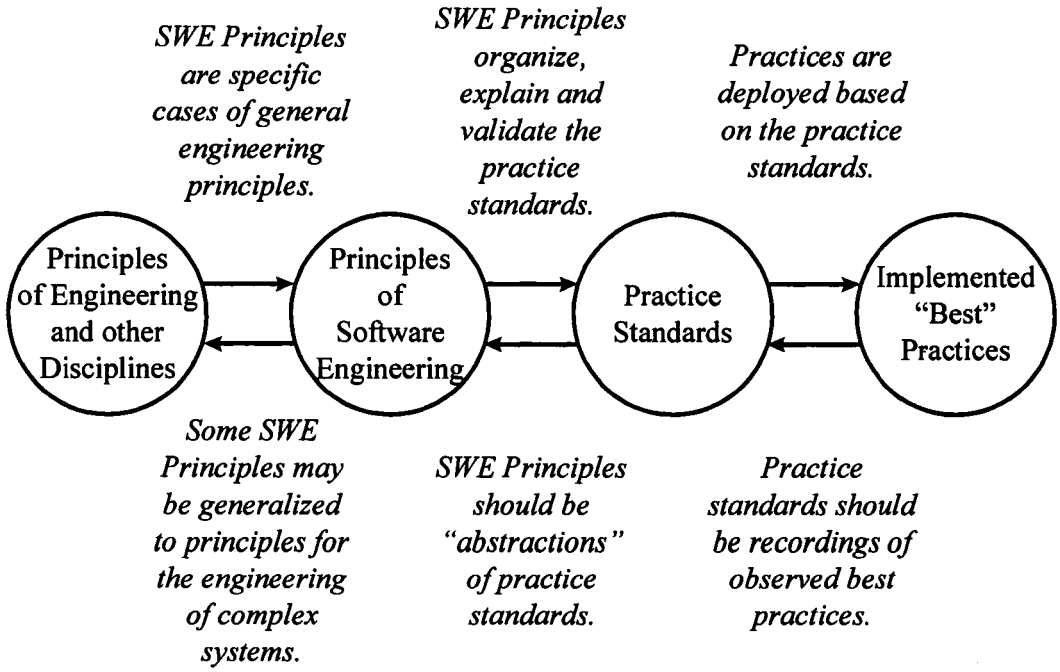

Figure 3 Modèle du rôle des principes fondamentaux (Bourque et al. 2002)

Ainsi, les organismes gérant le corpus des normes du génie logiciel ont un urgent besoin des fondements de la discipline pour baliser et supporter le corpus des normes. À ce

sujet, nous citons les propos de James Moore (1998) : *« Perhaps a more basic requirement is for the identification of a set of fundamental principles that would serve to explain and motivate the provisions of the various standards. » (p.5)*

Les principes comme base du curriculum du génie logiciel

Davis (1998) souligne qu'une discipline du génie est fondamentalement définie par son curriculum. La définition du curriculum est importante et se doit d'être rigoureuse. Les programmes du génie sont accrédités sur le curriculum qui définit les connaissances que le futur ingénieur doit connaître. Meyer (2001) propose que le curriculum du génie logiciel comporte cinq éléments importants, dont le premier serait les principes. Ainsi, nous citons : *« Among the most important things that professional software engineer know is a set of concepts[2] that recur throughout their work » (p.29)*

De plus, Meyer (2001) ajoute que les principes ou concepts fondamentaux sont : « *...they [principles] make up the most important, if sometimes immaterial, body of knowledge that, as a teacher, I try to impart to novices » (p.29)*

Meyer (2001) reconnaît l'importance et la place des principes fondamentaux dans le curriculum du génie logiciel. Cependant, la discipline ne peut pas encore compter sur un noyau de principes fondamentaux reconnus par la communauté.

Les principes comme base au SWEBOK

L'équipe de recherche du laboratoire de recherche en génie du logiciel[3], a entrepris depuis 1998 d'établir le corpus des connaissances du génie logiciel (SWEBOK[4]). Le

---

[2] Meyer définit le terme *principe* comme suit : « *lasting concepts that underlies the whole field* »
[3] GELOG : Laboratoire de recherche en génie du logiciel (www.gelog.etsmtl.ca)

contenu du corpus fait l'objet d'un consensus d'envergure internationale. Ainsi, ce guide sert déjà à des organismes pour définir les bases du curriculum en génie logiciel ou pour définir les activités et les tâches d'un ingénieur logiciel. Actuellement, le corpus n'est pas appuyé explicitement sur un ensemble de principes fondamentaux de la discipline. D'une façon implicite, on peut retrouver certains principes identifiés par Bourque et al. (2002), comme en fait mention Séguin (2001) dans un rapport technique sur l'étude particulière d'un principe fondamental par rapport à deux chapitres du SWEBOK.

L'établissement d'un corpus de principes fondamentaux du génie logiciel fournirait un cadre d'analyse pour SWEBOK et permettrait de le raffiner (Bourque et al. 2002).

Principes, base du génie logiciel ?

Les principes fondamentaux seraient à la base de plusieurs composants de la discipline, tel que résumé dans la figure suivante :

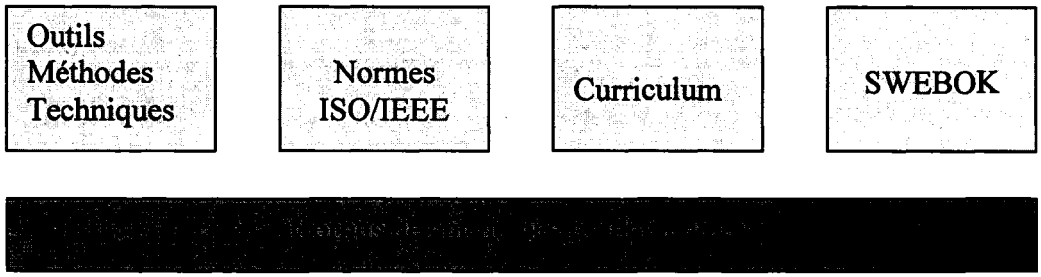

Figure 4 Principes fondamentaux comme assises de la discipline

Compte tenu que la discipline du génie logiciel ne repose pas sur des lois de la physique, l'identification des principes n'est pas une tâche facile. Comme nous allons le présenter à la section suivante, plusieurs auteurs ont écrit sur le sujet. Il est donc impératif pour

---

[4] SWEBOK : Software Engineering Body of Knowledge

déterminer le cadre de cette recherche, de présenter une synthèse des travaux antérieurs concernant les principes du génie logiciel. À la suite de cette synthèse, les objectifs et la méthodologie de recherche seront présentés au chapitre 2.

Le présent document est composé de sept chapitres. Au chapitre 1, nous présentons une revue de littérature sur les principes fondamentaux. À la suite de cette revue de littérature, nous présentons au chapitre 2 le but et la motivation de cette thèse, les objectifs à atteindre et la méthodologie de recherche.

Le chapitre 3 définit le cadre conceptuel et ses principaux éléments.

Le chapitre 4 présente l'évaluation des principes répertoriés en appliquant les critères d'identification individuels.

Le chapitre 5 procède à l'évaluation des principes retenus au chapitre 4, les catégorise, applique les critères d'identification d'ensemble et effectue des liens avec les processus de la norme ISO/IEC 12207.

Le chapitre 6 procède à une évaluation du degré de couverture des principes retenus avec, d'une part, un modèle d'ingénierie (Moore 2006) et, d'autre part, avec le corpus des normes du génie logiciel de l'IEEE.

Le chapitre 7 présente une synthèse de la recherche effectuée.

CHAPITRE 1

L'ÉTAT DE L'ART SUR LES PRINCIPES DU GÉNIE LOGICIEL

1.1 Synthèse de l'état de l'art sur les principes

Dans cette section, nous exposons une synthèse des travaux portant explicitement sur les principes du génie logiciel depuis le début des années 1970. Cette revue contient la vision individuelle de dix auteurs et la vision collective de cinq groupes d'auteurs. La présentation des contributions des différents auteurs se fait en ordre chronologique afin de montrer l'évolution de la recherche sur le thème des principes. Il est à noter que les principes présentés par les auteurs ne seront pas traduits en français afin de préserver toute leur signification originale.

Winston Royce (1970) propose sa vision concernant la gestion de grands projets de développement logiciel. Il définit cinq règles qu'il nomme « étapes » à suivre. Ces règles sont principalement basées sur l'expérience professionnelle de l'auteur dans l'application du cycle de vie en cascades. L'auteur souligne que ce modèle comporte des risques pour les grands projets de développement. Il propose donc cinq « étapes » pour l'améliorer et diminuer les risques en cours de projet.

Tableau I

Étapes de Royce (1970)

1. Program design comes first 2. Documentation must be current and complete 3. Do the job twice if possible 4. Testing must be planned, controlled and monitored 5. Involve the customer

Royce souligne que l'application de ces règles peut être un facteur de succès pour les grands projets de développement. Même si l'auteur ne fait pas mention du terme « principe », il formule des règles à suivre dont certaines pourraient être à la source de principes du génie logiciel.

Ross, Goodenough et Irvine (1975) est l'un des premiers groupes d'auteurs à reconnaître de façon explicite que les outils et les méthodes ne sont pas suffisants pour développer du logiciel. Ainsi, il y aurait aussi des principes de base à appliquer au sein du processus de développement. Les auteurs présentent un cadre conceptuel à trois dimensions destiné à mieux intégrer les concepts de base du génie logiciel. Ce cadre inclut les phases du cycle de développement, les propriétés (qualités) désirables du logiciel et les principes. Les propriétés ou qualités du logiciel sont : « *modifiability, efficiency, reliability, understandability* ». Les auteurs affirment que ces propriétés font l'objet d'un consensus auprès de la communauté du génie logiciel. Les auteurs identifient sept principes, explicitement nommés comme étant « les » principes du génie logiciel. Ces principes sont : « *modularity, abstraction, hiding, localization, uniformity, completeness, comfirmability* »[5]. Les auteurs ne précisent pas la provenance des principes identifiés, mais affirment que ceux-ci ont déjà été identifiés par les « *observateurs* » de la discipline, sans donner plus détails.

Ross et al. (1975) ont commenté chacun des énoncés de principe identifiés afin que le lecteur puisse en saisir le sens et la portée. Cependant, aucune définition du terme principe n'est donnée, ni de critères d'identification de ceux-ci. Malgré l'originalité du cadre présenté, celui-ci manque de détails fins concernant les liens entre les éléments du cadre présenté. Ainsi, le modèle reste muet sur les principes qui aideraient à satisfaire une propriété en particulier, comme le feront, près de 20 ans plus tard, Booch et Bryan (1994).

---

[5] Nous conservons la nomination anglaise afin d'éviter les ambiguïtés de traduction.

Harlan Mills (1980) publie en 1980 un article sur les principes du génie logiciel. Malgré son titre prometteur, l'auteur n'aborde que très peu la question des principes en se concentrant plutôt sur la crise du logiciel et sur les caractéristiques de base du génie logiciel. Mills (1980) définit le génie logiciel comme étant « *a growing set of disciplines* » qui se subdiviseraient en trois catégories : design, développement et management. Mills identifie sommairement deux principes au niveau du design soit la « *décomposition modulaire* » et la « *programmation structurée* ». L'auteur ne propose pas de méthodologie pour aborder le thème des principes, ni de définition et de critères d'identification des principes.

Les travaux de **Manny Lehman (1980)** portent sur l'évolution des grands systèmes informatiques. Lehman a analysé différentes données historiques provenant de projets de maintenance sur une période de sept ans. De ses analyses, Lehman a observé certaines constantes qu'il associe à des « lois » ou à des principes de base de l'évolution des systèmes. L'auteur affirme que ces « lois » se rapprocheraient des lois de la physique ou de la biologie. Lehman propose les lois suivantes :

Tableau II

Lois de l'évolution des systèmes logiciels (Lehman 1980)

1. The law of continuing change
2. The law of increasing complexity
3. The fundamental law
4. The law of organizational stability
5. The law of conservation of familiarity

Lehman observe que l'évolution du logiciel est directement influencée par le jugement des personnes impliquées. De ce fait, Lehman souligne qu'il ne faut pas espérer découvrir des lois qui offriraient le même niveau de précision et de prévisibilité que celles de la physique, non influencées par le jugement. L'auteur souligne également

l'importance de développer une compréhension globale des principes à la base du génie logiciel et en particulier pour la phase de maintenance.

Bary Boehm (1983) se penche sur l'identification des principes du génie logiciel en analysant les données historiques de plusieurs projets réalisés par la firme TRW Defence. Les projets examinés représentent près de 30,000,000 d'heures personnes. De cette analyse, Boehm identifie sept principes indépendants qui seraient à la base de la discipline.

Boehm décrit chacun des énoncés de principe afin que le lecteur puisse en percevoir le sens et la portée. Boehm est le premier chercheur de notre revue à définir deux critères d'identification des principes. En premier lieu, les principes doivent être indépendants; la combinaison de deux principes n'engendre pas un troisième. En second lieu, les principes devraient couvrir l'ensemble des activités de la discipline. Même si Boehm ne définit pas explicitement le terme principe, nous observons que chacun des énoncés de principes proposés est formulé sous forme d'une règle d'action.

Tableau III

Principes proposés par Boehm (1983)

1. Manage using phased life cycle plan
2. Perform continuous validation
3. Maintain disciplined product control
4. Use modern programming practices
5. Maintain clear accountability for results
6. Use better and fewer people
7. Maintain a commitment to improve the process

L'auteur souligne que l'application de ces sept principes ne garantit pas d'une façon absolue le succès d'un projet à cause de la complexité du développement logiciel. De plus, Il souligne que la discipline n'est pas encore complètement comprise et qu'il y a

encore beaucoup de place laissée au jugement des individus et ce même dans l'interprétation de la signification des principes proposés. Cependant, Boehm mentionne que la prise en compte des principes proposés permettrait d'éviter plusieurs erreurs classiques et d'identifier plus rapidement les difficultés ayant un grand impact sur le projet.

Les travaux de Boehm sont un point marquant dans l'étude des principes du génie logiciel. L'auteur présente une méthodologie en proposant des tableaux dont certains sont composés de données historiques[6] tirées de projets réels. De plus les énoncés de principes présentés sont formulés sous forme de règles ou prescriptions, tout en étant suffisamment commentés pour en percevoir le sens donné par l'auteur.

Booch et Bryan (1994) dans leur ouvrage « Software Engineering with Ada » dédient un chapitre sur les objectifs et les principes du génie logiciel. Les auteurs présentent une définition du génie logiciel qui intègre l'application des principes : *« is the application of sound engineering **principles** to the development of systems that are modifiable, efficient, reliable, and understandable » (p.32).*

Pour atteindre les critères de qualité nommés dans cette définition, les auteurs soulignent qu'il est essentiel d'appliquer des principes de base du génie logiciel intégrés à un processus de développement systématique et discipliné. Cependant, ils soulignent que la seule connaissance des principes n'est pas suffisante pour développer un produit satisfaisant aux critères de qualité mentionnés. Le processus doit aussi s'appuyer sur des méthodes. Les principes présentés par les auteurs sont les mêmes que ceux présentés par Ross et al. (1975). Cependant, Booch et Bryan ont précisé les relations entre les principes et les critères de qualité du logiciel

---

[6] Mesures sur, entre autres, les erreurs détectées, le nombre de personnes, de modules et la gestion de projet.

Tableau IV

Principes de Booch et al. (1994) en relation avec les critères de qualité[7]

	Propriétés désirables du logiciel			
Principes	Modifiability	Efficiency	Reliability	Understandability
Abstraction	X	X	X	X
Information hiding	X	X	X	X
Modularity	X		X	X
Localization	X		X	X
Uniformity				X
Completeness	X	X	X	
Confirmability	X	X	X	

Alan Davis (1995) a consacré un ouvrage complet sur les principes du développement logiciel. Son ouvrage est présenté comme un guide à l'intention des ingénieurs, des gestionnaires, des étudiants et des chercheurs en génie logiciel. Davis souligne que malgré une grande quantité d'articles et d'ouvrages portant sur les méthodes, les techniques et les outils du génie logiciel, très peu ont porté sur les principes à la base du génie logiciel. Davis est le premier auteur de notre revue de littérature à définir explicitement le terme principe comme étant : *« is a basic truth, rule or assumption about software engineering that holds regardless of the technique, tool or language selected »* (p. xi).

De plus, il ajoute que si le génie logiciel est réellement une discipline du génie, sa définition devrait être : *« It is the intelligent application of proven principles, techniques,*

[7] Adaptation faite à partir du texte des auteurs

languages and tools to the cost-effective creation and maintenance of software that satisfies user's needs » (p. x).[8]

Cette définition inclut explicitement les principes tout comme celle proposée par Booch et Bryan (1994).

Davis (1995), en référence aux travaux de Lehman (1980), souligne qu'à cause de la nature conceptuelle du logiciel, les lois de la physique ne peuvent s'appliquer. De ce fait, Davis suggère que le génie logiciel développe ses propres principes de base par l'observation de milliers de projets.

Davis propose un modèle de l'influence et du rôle des principes sur les techniques, les langages et les outils utilisés pour le développement de logiciel (Figure 5).

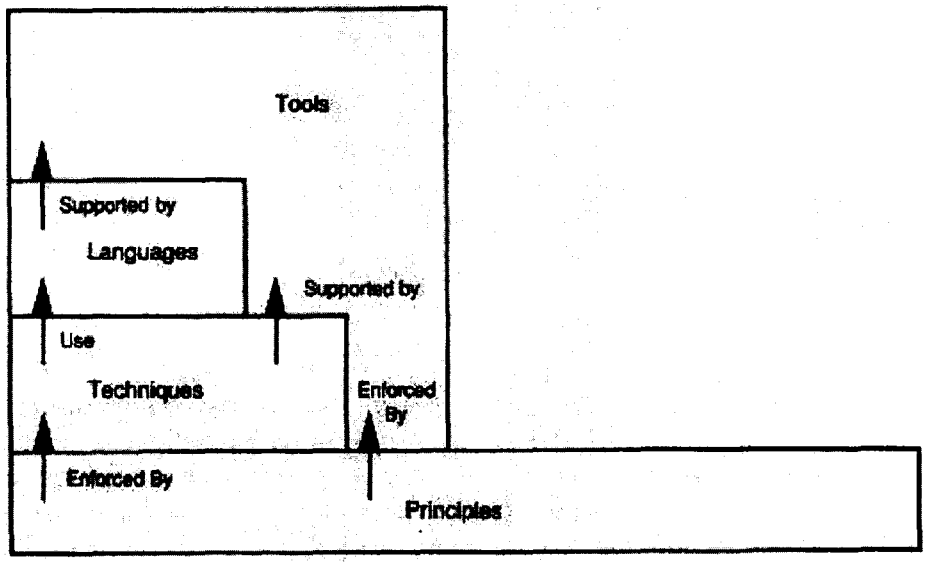

Figure 5 Influence des principes (Davis 1995 p. 4)

[8] Il est à noter que cette définition du génie logiciel est l'une des rares à inclure les besoins du client.

Davis propose 201 énoncés de principes et les regroupe selon huit catégories recoupant les activités du développement et des activités de support. Dans un article prépublication de son livre, Davis (1994) identifie parmi les 201 principes, les 15 plus importants.

Tableau V

Les 15 principes les plus importants (Davis 1994)

1. Make Quality Better	9. Put technique before tools
2. High-Quality Software is possible	10. Get it right before you make it faster
3. Give products to customers early	11. Inspect code
4. Determine the problem before writing the requirements	12. Good management is more important than good technology
5. Evaluate design alternatives	13. People are the key to success
6. Use an appropriate process mode	14. Follow with care
7. Use different languages for different phases	15. Take responsibility
8. Minimize intellectual distance	

Au niveau méthodologique, Davis fonde ses propos sur les travaux de Royce (1970), Lehman (1980) et Boehm (1983). Chacun des principes présentés a une référence directe à une publication, source principale de ses propos. En tout, Davis a compilé 124 références provenant de publications de praticiens et de chercheurs en génie logiciel pour soutenir ses 201 énoncés de principe. L'auteur n'a pas fait d'analyse détaillée pour tenter de généraliser les énoncés de principes pour en diminuer le nombre tel que fait par Boehm (1983) et Bourque et al. (2002). Également, l'auteur mentionne explicitement que les principes ne sont pas indépendants comme ceux de Boehm (1983) et que certains peuvent même être contradictoires selon le contexte. Même si le terme principe est défini, l'auteur ne propose pas de critères d'identification des principes. De plus, il ne fait pas mention de sa démarche méthodologique pour l'identification et la sélection des énoncés de principe.

Buschmann et al. (1996) dans leur ouvrage sur l'architecture du logiciel, affirment que la construction du logiciel est basée sur plusieurs principes fondamentaux. Les auteurs

ont identifié onze principes qui seraient, selon eux, les plus importants pour le volet architecture du logiciel.

Tableau VI

Principes proposés par Buschmann et al (1996)

1. Abstraction 2. Encapsulation 3. Information hiding 4. Modularization 5. Separation of concerns 6. Coupling and cohesion 7. Sufficiency, completeness and primitiveness	8. Separation of policy and implementation 9. Separation of interface and implementation 10. Single point of reference 11. Divide and conquer (decomposition)

Les auteurs ne définissent pas le terme principe, ni de critères d'identification. De plus, les auteurs ne nous indiquent pas en quoi ces principes seraient fondamentaux. Ils signalent seulement qu'ils ont fait un choix, mais sans expliquer leur démarche, ni identifier les critères de sélection utilisés. Ils ne présentent pas de modèle sur la place et le rôle des principes. Ils considèrent comme étant synonyme les termes *principe* et *technique.* Enfin, ils affirment que ces principes sont : *« ...widely-accepted principles » (p. 397).*

Karl Wiegers (1996) propose la vision que l'amélioration du processus de développement et des produits passe d'abord par des changements dans la culture génie logiciel des équipes de développement et des organisations. Wiegers identifie 14 principes qui auraient une influence sur la culture du génie logiciel et implicitement sur la qualité et l'efficacité du processus de développement.

Tableau VII

Principes « culturels » proposés par Wiegers (1996)

1. Never let your boss or your customer talk you into doing a bad job
2. People need to feel the work they do is appreciated
3. Ongoing education is every team member's responsibility
4. Customer involvement is the most critical factor in software quality
5. Your greatest challenge is sharing the vision of the final product with the customer
6. Continual improvement of your software development process is both possible and essential
7. Written software development procedures can help build a shared culture of best practices
8. Quality is the top priority; long term productivity is a natural consequence of high quality
9. Strive to have a peer, rather than a customer, find a defect
10. A hey to software quality is to iterate many times on all development steps except coding: do this once.
11. Managing bug reports and change request is essential to controlling quality and maintenance
12. If you measure what you do, you can learn to do it better
13. Do what makes sense; don't resort to dogma
14. You can't change everything at once. Identify those changes that will yield the greatest benefits and begin to implement them next Monday

Malgré le fait que le texte de Wiegers soit appuyé par plus d'une centaine de références, l'auteur ne précise pas la démarche de sélection des principes, ni les sources de ceux-ci. Nous pouvons noter que l'auteur indique avoir eu l'occasion d'expérimenter dans un contexte industriel les principes proposés. L'auteur signale que les principes proposés sont aussi la base de la sélection des pratiques de développement pour améliorer le processus et les produits. Ainsi, les principes seraient la fondation de l'amélioration des processus et des produits.

Anthony Wasserman (1996) présente une liste comportant huit « concepts fondamentaux » du génie logiciel.

Tableau VIII

Concepts fondamentaux du génie logiciel proposés par Wasserman (1996)

1. Abstraction ▪ Incluant « information hiding » 2. Analysis and design methods and notation 3. User interface prototyping	4. Modularity and architecture ▪ Separation of concerns ▪ Localization ▪ Decomposition ▪ Design patterns 5. Software life cycle and process 6. Reuse 7. Metrics 8. Tools and integrated environment

Wasserman précise que ces concepts fondamentaux ont des liens entre eux. Ainsi ils ne seraient pas indépendants au sens donné par Boehm (1983). L'auteur souligne que ces concepts fondamentaux sont au cœur même des meilleures pratiques de l'industrie. Cependant, l'auteur ne définit pas précisément ce qu'il entend par « concepts fondamentaux ». Il ne précise pas non plus de critères d'identification et de sélection de ces concepts. De plus, il ne fait pas mention de la démarche méthodologique utilisée pour arriver à sa liste de concepts fondamentaux. L'absence d'une définition du terme « concept » par Wasserman, nous amène à constater également une certaine confusion entre les termes : *concept, technique, notion, outil* et *méthode* au sein du texte et des exemples donnés.

Tom Maibaum (2000) s'intéresse aux fondements mathématiques du génie logiciel. Les prémisses de l'auteur sont : si le génie logiciel est une discipline du génie, celle-ci devrait avoir des sources mathématiques tout comme les autres disciplines classiques. L'auteur définit le génie logiciel comme suit : *« Software engineering is the activity of systematically constructing software based systems which are fit for purpose » (p.163).*

Maibaum aborde le fait que le design en génie logiciel serait encore plutôt du type radical; i.e. un design ou une création sur mesure pour chacun des logiciels développés.

Se basant sur les travaux de Vincenti (1990), l'auteur est d'avis que le génie logiciel devrait suivre les principes du design normal en utilisant des composants logiciels déjà faits pour assembler de nouveaux logiciels.

Au niveau des principes, Maibaum aborde le principe de la modularité en soulignant que c'est la seule « méthode » efficace pour aborder la complexité. Il ajoute que ce principe ne devrait pas seulement se retrouver au niveau des langages de programmation, mais aussi au niveau des langages de design (notation) et de spécifications. Par la suite, l'auteur mentionne que le génie logiciel a de la difficulté à adopter des principes de mesures[9] qui sont mieux intégrés dans les processus des autres disciplines du génie.

L'auteur n'oriente pas principalement ses propos sur les principes du génie logiciel. De plus, il ne définit pas le terme principe, ni la notion de « fondation » du génie logiciel. L'auteur n'appuie pas ses propos sur une méthodologie particulièrement explicite.

Paul Taylor (2001) utilise les principes de design de Mayall (1979) comme cadre conceptuel descriptif dans le but d'interpréter ces principes en rapport aux activités de design du génie logiciel. Mayall a décrit dix principes de design dans un texte de nature académique destiné aux étudiants en design. Mayall fonde ses principes sur les pensées contemporaines d'auteurs des années 60 et 70 concernant les fondements du design. Le tableau IX présente les dix principes de Mayall.

---

[9] « ...adopting engineering principles of measurement... » P. 171.

Tableau IX

Principes de design de Mayall (Taylor 2001)

1. Totality – all artefact characteristics are interrelated
2. Time – the features and characteristics of all products change as time pass
3. Value – the characteristics of all products have different relative value
4. Resources – the design, manufacture and life of all products and systems depend upon the materials, tools and skills
5. Synthesis – design as resolution of conflicting forces
6. Iteration
7. Change
8. Roles and Relationships
9. Competence
10. Service

Taylor souligne que ces principes représentent un cadre descriptif basé sur les opinions de Mayall et de ceux en référence de ses travaux. Taylor tente d'interpréter ces principes en rapport aux activités de design logiciel. L'auteur souligne des corrélations fortes entre les principes de Mayall et le design logiciel, malgré les différences entre les produits issus du design : physique vs conceptuel (logiciel).

Bertrand Meyer (2001) souligne qu'il n'y a pas encore une définition du génie logiciel qui fasse l'objet d'un consensus universel. Cependant, les institutions qui enseignent le génie logiciel (informatique ou génie) ont la responsabilité de former, aujourd'hui, des professionnels en mesure de développer et de maintenir des logiciels à la satisfaction des utilisateurs. Meyer suggère un curriculum composé de cinq composants dont le premier est les principes. Meyer (2001) définit le terme principe comme suit :

- « *Principle : lasting concept that underlie the whole field* »
- « *... is a set of creative concepts* » *(p.29)*

L'auteur présente et commente 13 principes ou concepts fondamentaux qui représentent les fondements de la connaissance qu'un professionnel en logiciel doit savoir.

Tableau X

Principes proposés par Meyer (2001)

1. Abstraction 2. Distinction between specification and implementation 3. Recursion 4. Information hiding 5. Reuse 6. Battling complexity	7. Scaling up 8. Design for change 9. Classification 10. Typing 11. Contracts 12. Exception handling 13. Errors and debugging

Malgré le fait que l'auteur définisse le terme principe, nous constatons une certaine ambiguïté dans l'utilisation des termes *principe* et *concept;* termes qui n'ont pas la même signification. De ce fait, nous pouvons constater cette confusion dans la liste des 13 principes présentés. Ainsi, le principe no.8 « Design for change » est plus près d'une règle à suivre que le principe no.10 « Typing ». Nous constatons que certains des principes présentés seraient plutôt des caractéristiques recherchées dans un langage de programmation[10] (exemples : exception handling, recursion et typing).

En 1996, **Bourque, Dupuis, Abran, Moore, Tripp et Wolf (2002)** entreprennent une recherche empirique sur l'identification des principes fondamentaux du génie logiciel. Les prémisses de recherche de l'équipe étaient, entre autres, les suivantes :

- La discipline ne dispose pas de principes fondamentaux universellement reconnus;
- Difficultés de différencier les activités du génie logiciel de celles de l'informatique;
- Les besoins des organismes de normes;
- La difficulté de définir le curriculum de la discipline.

[10] Ces « principes » sont intégrés au langage Eiffel dont l'auteur est le promoteur.

Les auteurs ont accordé une grande importance à la conception de la démarche méthodologique de la recherche et à bien la documenter, ce que l'on ne retrouve pas, en général, au sein des travaux antérieurs sur le sujet. La figure 6 illustre la démarche méthodologique de cette recherche de Bourque et al (2002).

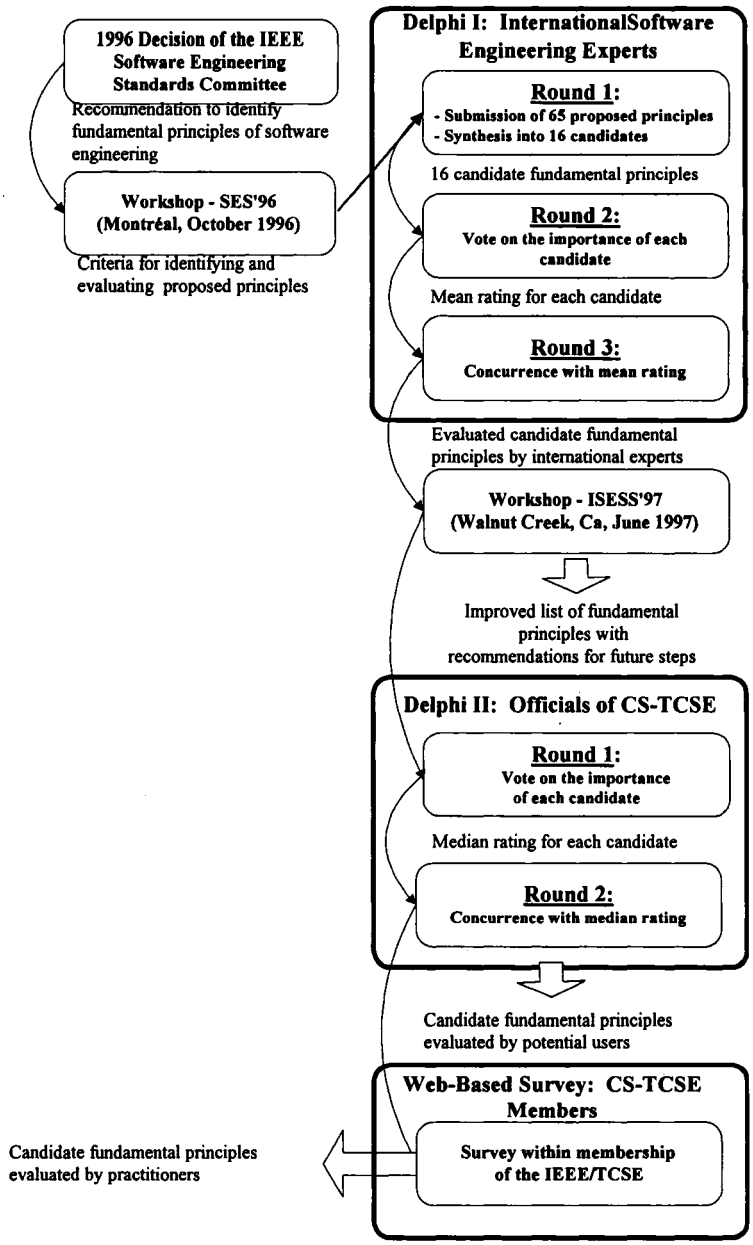

Figure 6 Démarche méthodologique de l'étude de Bourque et al. (2002)

La définition choisie du génie logiciel est celle adoptée par l'IEEE (IEEE 610.12).

> *"(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e. the application of engineering to software.*
>
> *(2) The study of approaches as in (1). "*

Bourque et al. (2002) identifient huit critères permettant d'identifier et d'évaluer les énoncés de principes.

Tableau XI

Critères d'identification des principes (Bourque et al. (2002))

1. Les principes fondamentaux sont moins spécifiques que les méthodes et technologies;
2. Les principes fondamentaux sont plus durables que les méthodes et techniques;
3. Les principes fondamentaux sont identifiés à partir des pratiques et devraient avoir un lien avec les meilleures pratiques;
4. Les principes fondamentaux du génie logiciel ne doivent pas contredire des principes fondamentaux plus généraux;
5. Les principes ne doivent pas cacher un compromis;
6. Il peut y avoir cependant un compromis dans l'application d'un principe;
7. Les principes doivent être assez précis pour être supportés ou contredits par l'expérimentation;
8. Un principe doit être relié à un ou plusieurs concepts sous-jacents.

Un principe est défini comme suit :

> *« By contrast [to concepts], fundamental principles are to be regarded as engineering statements which prescribe constraints on solutions to problems or constraints on process of developing solutions » (p.2).*

Bourque et al. (2002) fournissent une définition du terme principe. Cependant, cette définition ne fait pas explicitement mention du terme « règle » tel que spécifié dans la définition donnée par Davis (1995). Les termes « *constraints* » et « *prescribe* » nous

suggèrent implicitement un concept de règle. La définition aussi fait mention du terme processus, mais implicitement du produit par l'entremise du terme « solution ». Ainsi, on peut y retrouver deux ingrédients importants du génie logiciel, soit le processus et le produit. Cependant, les auteurs ne définissent pas ce qu'ils entendent par le terme « fondamental », comme c'est aussi le cas dans la plupart des travaux antérieurs.

Suite à deux ateliers de travail, deux études Delphi et un sondage Web, Bourque et al. ont obtenu une liste de 15 énoncés de principes fondamentaux dits candidats. Il est à noter que les activités d'élucidation, d'analyse et de mesure du consensus ont impliqué près de 600 personnes[11] (chercheurs, professeurs, professionnels), ce qui fait de cette recherche empirique, l'une des plus importantes à ce jour sur le thème des principes du génie logiciel. La liste des principes candidats identifiés est présentée au tableau XII en ordre alphabétique.

Tableau XII

Liste des 15 principes candidats fondamentaux (Bourque et al. 2002)

A. Apply and use quantitative measurements in decision making
B. Build with and for reuse
C. Control complexity with multiple perspectives and multiple levels of abstraction
D. Define software artifacts rigorously
E. Establish a software process that provides flexibility
F. Implement a disciplined approach and improve it continuously
G. Invest in the understanding of the problem
H. Manage quality throughout the life cycle as formally as possible
I. Minimize software component interaction
J. Produce software in a stepwise fashion
K. Set quality objectives for each deliverable product
L. Since change is inherent to software, plan for it and manage it
M. Since tradeoffs are inherent to software engineering, make them explicit and document it
N. To improve design, study previous solutions to similar problems.
O. Uncertainty is unavoidable in software engineering. Identify and manage it.

[11] Incluant des auteurs bien connus de la discipline tels Brooks, Ghezzi, Gilb, Pressman.

Bourque et al. (2002) ont identifié chacun des principes-candidats à l'aide d'une lettre au lieu d'un nombre, ceci afin d'éviter toute référence à un ordre ou une priorité dans les principes proposés.

À l'aide des participants, l'équipe de recherche a été en mesure d'évaluer le degré de consensus de chacun des énoncés de principes. Également, une description sommaire des énoncés a aussi été faite afin de guider les participants et les lecteurs dans la compréhension de chacun des énoncés de principes. Même si les énoncés de principes présentés sont basés sur l'opinion des participants, cette recherche offre la vision d'un groupe et non l'opinion individuelle d'un auteur, comme c'est le cas de plusieurs travaux antérieurs recensés sur le sujet.

Ghezzi et al. (2003) sont les auteurs d'un livre de référence sur les fondements du génie logiciel. La particularité de cet ouvrage est de présenter les principes fondamentaux du génie logiciel et par la suite, de montrer aux lecteurs comment ceux-ci peuvent être appliqués aux différentes phases du processus de développement. Les auteurs affirment que l'application des principes est essentielle au développement du logiciel : « *...principles that we believe are essential to the multi-person construction of multi-version software* » *(p.2).*

Ghezzi et al. ajoutent que les principes sont plus importants que les notions ou les méthodes, en soulignant que leur apprentissage permet aux ingénieurs logiciels d'évaluer la meilleure méthode à utiliser selon le contexte propre d'un projet. Également, ils soulignent que les principes sont généraux, donc plus durables au passage du temps et des révolutions technologiques. Les auteurs définissent le terme principe comme suit : « *...they are general and abstract statements describing desirable proprieties of software process and products* » *(p.41).*

Il est à noter que les auteurs distinguent explicitement dans la définition le produit et le processus. Ils signalent également que les principes à eux seuls ne sont pas suffisants pour assurer le développement du logiciel; l'ingénieur aura aussi besoin de méthodes et de techniques pour appliquer les principes. Les auteurs présentent sept principes, tableau XIII.

Tableau XIII

Principes proposés par Ghezzi et al. 2003

1. Rigor and formality 2. Separation of concerns 3. Modularity 4. Abstraction	5. Anticipation of change 6. Generality 7. Incrementality

Ghezzi et al. soulignent qu'ils ont fait le choix des principes en considérant particulièrement deux propriétés désirables du logiciel; soit la fiabilité (« reliability ») et l'évolutionabilité (« evolvability », sous la catégorie de « maintainability »). Ainsi, ils mentionnent que : *« ...the choice of the principles and techniques is determined by software quality goals » (p.42).*

Les auteurs soutiennent les idées avancées par d'autres travaux antérieurs que les principes fondamentaux sont à la base du génie logiciel et qu'ils sont un fondement plus durable que les méthodes et les techniques : *« ...they [principles] constitute the foundation upon which all the rest may be built » (p.64).*

Les auteurs constatent la stabilité des principes, contrairement à Davis (1995) qui voyait plutôt une forme de renouvellement des principes dans le temps.[12] Ils présentent aussi un modèle qui a des similarités avec le modèle présenté par Davis (1995). Les auteurs décrivent chacun des principes en faisant certains liens avec d'autres et en indiquant leur

[12] Nous pensons que Davis confond à l'occasion « principe » et « pratique ».

relation soit avec le processus, le produit ou les deux. À ce sujet, Ghezzi et al. sont les premiers de notre revue de littérature à faire cette distinction.

Le texte de Ghezzi et al. est une étape importante[13] dans l'avancement des travaux sur les principes du génie logiciel. En plus de reconnaître leur importance, les auteurs fondent tout leur discours sur ceux-ci. Ainsi, les chapitres subséquents du livre décrivent l'applicabilité de ces principes dans les différentes phases du processus de développement.

1.2 Synthèse sur les travaux antérieurs

1.2.1 Vue d'ensemble

Notre revue de littérature remonte jusqu'à 1970, soit une couverture de plus de 30 ans. Un premier constat est que seulement un nombre restreint d'auteurs se sont penchés sur l'identification et la documentation des principes fondamentaux du génie logiciel, comparativement au nombre de travaux portant sur les outils, les techniques ou les méthodes. Nous constatons qu'un certain consensus se dégage chez les auteurs de notre revue à l'effet que les principes seraient à la base de la discipline du génie logiciel. Des auteurs tels Lehman (1980), Davis (1995) et Ghezzi et al. (2003) font un parallèle avec les disciplines classiques du génie qui s'appuient sur des principes ou des lois de la physique, de la chimie ou de la biologie pour l'établissement des bases de ces disciplines. Cependant, Davis (1995) souligne, qu'à cause de la nature particulière du génie logiciel, celui-ci devra développer ses propres principes. Lehman (1980) ajoute qu'à cause de l'implication omniprésente du jugement des individus dans le processus de développement, les principes du génie logiciel ne seraient pas aussi précis et prévisibles que ceux de la physique, par exemple, et sujets à l'interprétation.

---

[13] Tout comme les travaux de Bourque et al., et de Boehm.

L'évolution rapide des technologies associée à un cycle de vie très court de celles-ci rend rapidement désuet les outils, les techniques et certaines méthodes. Ainsi, la définition d'un noyau plus stable et indépendant des techniques serait souhaitable (Davis (1995) et Wasserman (1996)). Ghezzi et al. (2003) ajoutent que ce noyau serait aussi indépendant du cycle de vie choisi pour réaliser un projet de développement logiciel.

1.2.2 Inventaire des termes et base méthodologique

Le tableau XIV dresse une première synthèse des travaux cités concernant le vocabulaire et les critères utilisés pour identifier les principes, du nombre d'énoncés identifiés et du type de formulation utilisé par les auteurs. Également, il y est indiqué si l'auteur a donné ou non une définition explicite du terme « principe » et de la méthodologie de recherche des principes identifiés.

Tableau XIV

Caractéristiques des publications sur les principes du génie logiciel

Auteurs	Vocabulaire	Définition	Critères	Nombre	Formulation	Méthodologie
Royce (1970)	Étapes	Non	Non	5	Règle	Opinion
Ross et al (1975)	Principe	Non	Non	7	Concept	Littérature/ opinion
Lehman (1980)	Lois	Non	Non	5	Concept	Observation/ analyse
Mills (1980)	Principe	Non	Non	4	Concept	Opinion
Boehm (1983)	Principe	Non	2	7	Règle	Observation
Booch et al. (1994)	Principe	Non	Non	7	Concept	Littérature
Davis (1995)	Principe	Oui	Non	201	Règle	Littérature
Buschman et al. (1996)	Principe/ Technique	Non	Non	10	Concept	Littérature

Tableau XIV (suite)

Auteurs	Vocabulaire	Définition	Critères	Nombre	Formulation	Méthodologie
Wasserman (19960	Concept	Non	Non	8	Concept	Opinion/ littérature
Wiegers (1996)	Principe	Non	Non	14	Règle	Observation/ opinion
Maibaum (2000)	Principe	Non	Non	3	Concept	Opinion
Meyer (2001)	Principe	Oui	Non	13	Mixte	Opinion
Taylor (2001)	Principe	Non	Non	10	Concept	Opinion
Bourque al. (2002)	Principe	Oui	8	15	Règle	Opinion de groupe
Ghezzi et al. (2003)	Principe	Oui	Non	7	Mixte	Littérature/ opinion

Ainsi, dix des treize auteurs utilisent le terme principe. Buschman et al (1996) emploient les termes *principe* et « enabling technique » comme étant des synonymes, tandis que Lehman (1980) utilise le terme « loi » d'évolution du logiciel. Malgré l'utilisation majoritaire du terme *principe*, seulement quatre auteurs le définissent explicitement. Cette lacune importante au niveau méthodologique pourrait expliquer la confusion observée dans l'utilisation comme synonymes des termes *principe*, *concept*, *technique* et *notion*.

La formulation des énoncés des principes varie d'un simple mot à une phrase indiquant une règle à suivre. L'absence fréquente de définitions claires pourrait être à l'origine du grand éventail d'énoncés de principe du génie logiciel, sans pour autant obtenir un consensus de la part de la communauté. De plus, deux auteurs ont utilisé le qualificatif « fondamental » sans mentionner de signification particulière. Nous observons que les auteurs ne partagent pas tous la même définition du génie logiciel et peuvent à l'occasion se concentrer sur une spécialité du génie logiciel, comme Buschman et al. (1996) sur l'architecture du logiciel.

Nous observons aussi que les différents principes recensés pourraient se catégoriser sous trois pôles. Ainsi, Bourque et al. (2002) et plus précisément Ghezzi et al. (2003) distinguent des principes axés *produit* et d'autres axés *processus*. Wiegers (1996) aborde des principes axés sur les individus impliqués dans le processus de développement.

1.2.3 Critères d'identification

Seulement deux auteurs ont identifié explicitement des critères pour identifier les principes. Boehm (1983) en a identifié deux et Bourque et al.(2002) huit. D'une façon implicite, Ross et al. (1975), Booch et al. (1994) et Ghezzi et al. (2003) ont fait un lien avec les caractéristiques de qualité du logiciel. On pourrait considérer ces caractéristiques comme étant implicitement des critères.

1.2.4 Formulation des énoncés

Les auteurs recensés ont utilisé différentes approches pour formuler les énoncés de principes. Ainsi, cinq auteurs ont formulé les énoncés comme des règles à suivre; sept autres utilisent une formulation sous forme de concepts, tandis que deux utilisent une formulation mixte. Cette différence pourrait être liée au manque de définition du terme principe constaté. À titre d'exemple, Bourque et al. (2002) et Davis (1995) définissent le terme principe comme une prescription ou une règle à suivre. Ainsi, leurs énoncés de principe sont donc formulés comme des règles à suivre. Parmi les auteurs qui ont préféré formuler les principes sous forme de concept, aucun n'a défini le terme principe dans leurs travaux.

1.2.5 Sources des principes

Pour plus de la moitié des travaux cités, les principes présentés représentent essentiellement l'opinion personnelle des auteurs. Seulement quatre auteurs appuient leurs propos sur des observations historiques tirées de projets réels. Bourque et

al. (2002) présentent la synthèse des opinions de plus de 600 individus ayant participés aux divers travaux d'identification des principes. Globalement, 62% des travaux sont basés sur l'opinion de l'auteur, 23% sur des références de la littérature et 15% sur des données historiques. Cette constatation pourrait être aussi l'une des raisons de la grande multitude des énoncés de principe recensés.

Le tableau XV présente une autre synthèse des travaux antérieurs. Ainsi, on y retrouve le type de publication, le niveau de la discussion, le type de méthodologie, le nombre de références supportant les propos de l'auteur et la portée des principes proposés.

Tableau XV

Classification des travaux revus

Auteurs	Type	Discussion	Méthodologie	Références	Portée
Royce (1970)	Article	Théorique	-	0	Cycle de vie
Ross et al. (1975)	Article	Théorique	Implicite/analytique	20	Cycle de vie
Mills (1980)	Article	Théorique	-	16	Général
Lehman (1980)	Article	Empirique	Implicite/observation	13	Maintenance
Boehm (1983)	Article	Empirique	Implicite	49	Cycle de vie
Booch/Bryan (1994)	Livre	Théorique	-	12	Construction
Davis (1995)	Livre	Théorique	Implicite/analytique	124	Cycle de vie
Buschmanal. (1996)	Livre	Théorique	-	10	Architecture
Wasserman (1996)	Article	Théorique	-	19	Général
Wiegers (1996)	Livre	Théorique	Expérimentation		Culture GL
Maibaum (2000)	Article	Théorique	-	11	Général
Meyer (2001)	Article	Théorique	-	10	Curriculum

Tableau XV (suite)

Auteurs	Type	Discussion	Méthodologie	Références	Portée
Taylor (2001)	Article	Théorique	Analytique	22	Design
Bourque al. (2002)	Article	Empirique	Explicite	11	Cycle de vie
Ghezzi al. (2003)	Livre	Théorique	Implicite	24	Cycle de vie

1.2.6 Méthodologie

Nous constatons que la démarche méthodologique est une faiblesse évidente pour la majorité des travaux répertoriés. Ainsi, plus de 50% des travaux ne comportent pas de démarche méthodologique documentée ou explicite. Cinq auteurs font référence à une démarche implicite, mais sans en préciser les détails. À titre d'exemple, Boehm (1983) affirme que ses travaux portent sur l'analyse de 30 millions d'heures de projets réels; cependant, il ne mentionne pas de quelle façon il a analysé cette quantité considérable d'heures. Seuls les travaux de Bourque et al. (2002) exposent un plan de recherche et une démarche méthodologique explicite et documentée sur le déroulement de leurs travaux.

CHAPITRE 2

OBJECTIFS ET MÉTHODOLOGIE DE RECHERCHE

2.1 Objectifs de la recherche

La revue des travaux antérieurs a répertorié plus de 300 principes du génie logiciel proposés à la communauté depuis 35 ans sans qu'aucune de ces listes ne fassent l'objet d'un consensus de la part de la communauté. De plus, les 2/3 des principes proposés proviennent d'opinions personnelles des auteurs et, dans la majorité des cas, la méthode pour identifier ces principes est inexistante. Nous constatons que même dans le cas de travaux récents tels ceux de Bourque et al. (2002) où l'on retrouve une méthode rigoureuse, les résultats obtenus ne remplacent pas les autres listes de principes, mais ils s'ajoutent aux propositions faites depuis 1970. Ainsi, nous constatons qu'il est maintenant difficile de poursuivre les travaux sur les principes en ne choisissant qu'une liste en particulier. Dans le but de faire progresser les travaux sur les principes fondamentaux du génie logiciel, il est maintenant impératif de traiter la quantité impressionnante des principes recensés depuis 35 ans en les analysant un à un à l'aide d'une méthode rigoureuse afin de ne conserver à la fin qu'une liste réduite de propositions répondants à des critères d'identification précis. Cette liste réduite servira par la suite de point de départ pour la poursuite des activités de recherche sur les principes. Ce travail de recherche analytique représente le but principal de cette thèse sur les principes fondamentaux du génie logiciel.

Au niveau des objectifs, nous avons l'objectif, en premier lieu, de répondre à la question : qu'est ce qu'un principe? L'absence fréquente de définition de ce terme dans les études antérieures est une lacune majeure qui a eu des conséquences marquées sur les principes proposés par les auteurs. En second lieu, nous avons l'objectif de définir des critères d'identification des principes afin de guider le processus d'analyse.

Nous avons aussi un troisième objectif de concevoir une méthodologie rigoureuse découpée en phases permettant une analyse transparente des principes recensés et d'offrir aux chercheurs une traçabilité sur les décisions prises en cours d'analyse. Ce niveau de traçabilité est fortement absente des travaux antérieurs. Pour atteindre ces objectifs, la méthodologie de recherche sera présentée à la section suivante.

Également, nous avons l'objectif de vérifier la liste des principes retenus afin d'évaluer le degré de couverture avec un modèle d'ingénierie (Moore 2006) et avec le corpus de normes du génie logiciel de l'IEEE.

La réalisation de ces objectifs, de même que la conception de la méthodologie de recherche, composent l'originalité de cette recherche par rapport aux travaux antérieurs sur les principes fondamentaux du génie logiciel.

Il n'est pas dans nos objectifs d'identifier de nouveaux principes, mais d'analyser les principes répertoriés.

2.2 Méthodologie de recherche

La méthodologie de recherche choisie est de nature analytique et est composée de quatre principales phases présentées au tableau XVI.

Tableau XVI

Phases de la méthodologie de recherche

Phases	Description
1) Définition du cadre conceptuel d'analyse	• Concepts du génie • Définition du génie logiciel et identification des concepts • Identification des activités du génie logiciel • Définitions : concept, principe, lois • Critères d'identification des principes
2) Évaluation selon les critères individuels	• Application des critères individuels
3) Évaluation selon les critères d'ensemble, catégorisation et liens avec la norme ISO/IEC12207	• Catégorisation des principes • Application des critères d'ensemble • Liens avec la norme ISO/IEC 12207
4) Évaluation du degré de couverture des principes retenus	• Avec les éléments d'un modèle de l'ingénierie • Avec les normes du génie logiciel de l'IEEE

2.2.1 Phase 1 - Définition du cadre conceptuel d'analyse

Afin de soutenir les phases d'analyse des principes recensés, nous devons définir le cadre conceptuel avec ses éléments. Le cadre conceptuel proposé est composé de six éléments de base, tel que présenté à la figure 7.

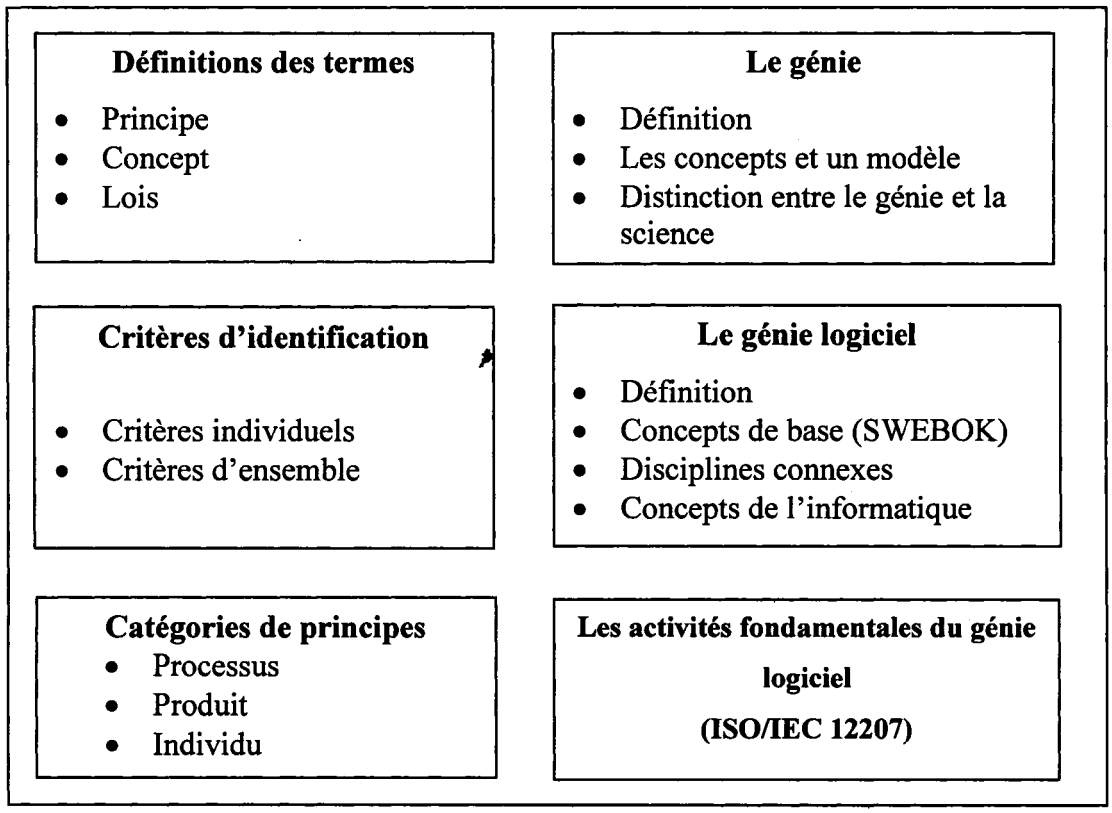

Figure 7 Éléments du cadre conceptuel

Le premier élément du cadre conceptuel est la définition des termes utilisés par les auteurs dans les travaux antérieurs. La question principale de recherche à répondre ici est : qu'est qu'un principe? La définition du terme principe est fondamentale pour la réalisation des travaux d'analyse des principes puisque certains critères d'identification en dépendent. Les termes concept et lois, termes souvent utilisés comme synonymes dans les travaux antérieurs, seront aussi définis afin de les distinguer par rapport au terme principe.

L'élément suivant du cadre conceptuel est la définition des critères d'identification d'un principe. Ces critères seront formulés en considérant les rares critères définis dans les

1. Les critères individuels seront utiles à filtrer, dans un premier temps à la phase 2, les principes considérés individuellement.

2. Les critères d'ensemble seront utilisés à la phase 3 pour filtrer le sous ensemble de principes issu de la phase 2.

L'élément catégorie de principes sera utilisé à la phase 3 afin de catégoriser les principes retenus à la phase 2 selon trois catégories. Lors de la revue de la littérature, nous avons constaté que certains auteurs avaient identifié des pôles ou des axes de regroupement des principes. Ghezzi et al. (2003) et Bourque et al. (2002) ont identifié le produit et les processus au sein même de leurs définitions[14] du terme *principe*. Wiegers (1996) a identifié les individus avec les aspects reliés à la culture du génie logiciel. Ainsi, nous proposons d'utiliser le modèle présenté à la figure 8 qui regroupe ces trois pôles.

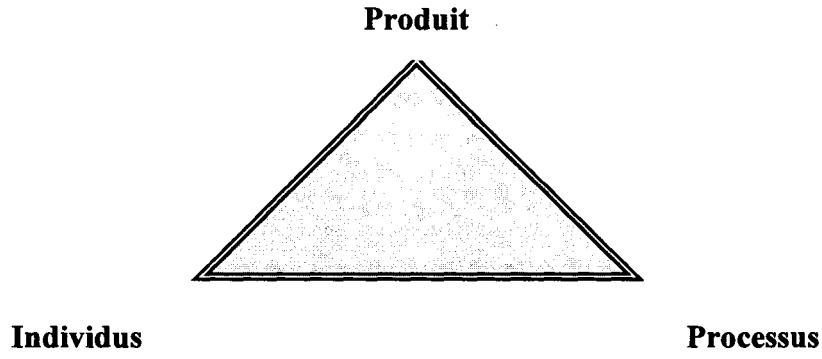

Figure 8 Pôles de regroupement des principes

Le génie logiciel est une discipline relativement nouvelle du génie, le génie est aussi un élément du cadre conceptuel. Ainsi, il est possible que des principes contiennent des concepts généraux du génie qui puissent s'appliquer aussi au génie logiciel. Afin d'être

[14] Bourque et al. font référence au terme solution que nous interprétons comme le produit.

en mesure d'identifier ces situations, les concepts de base du génie et ses caractéristiques seront identifiés. Le modèle d'ingénierie présenté par Moore (2006) sera aussi exposé.

Le génie logiciel est un élément incontournable du cadre conceptuel. Cet élément fournira une définition du génie logiciel, une identification des principaux concepts du génie logiciel et l'identification des disciplines frontières. De ces dernières, l'informatique sera traitée en détail afin d'en identifier les concepts propres à cette discipline. Ces concepts seront utilisés à la phase 2 de l'analyse afin de les identifier dans les principes. Ainsi, les principes comportant des concepts de l'informatique ne seront pas conservés comme principes du génie logiciel. L'élément du génie logiciel est principalement fondé sur le guide du corpus des connaissances du génie logiciel SWEBOK (2004) qui présente, entre autres, les principaux concepts du génie logiciel.

Pour compléter le cadre conceptuel, les activités fondamentales du génie logiciel sont incluses telles que définies par la norme parapluie ISO/IEC 12207. Dans les travaux antérieurs, il a été constaté que certains principes sont en fait que de simples activités du génie logiciel. Comme un principe n'est pas une activité, il est nécessaire d'identifier ce type de propositions. Ainsi, les processus et activités présentées par la norme ISO/IEC 12207 serviront de références afin d'identifier les propositions qui représentent des activités à faire.

Ces six éléments constituant la base du cadre conceptuel seront utilisés dans les différentes phases de la méthodologie de recherche qui traitera les 308 principes recensés lors de cette recherche.

2.2.2 Phase 2 - Évaluation selon les critères individuels

La deuxième phase consiste à évaluer chacun des 308 principes en fonction des critères individuels définis au sein du cadre conceptuel et des autres éléments requis par cette

2.2.2 Phase 2 - Évaluation selon les critères individuels

La deuxième phase consiste à évaluer chacun des 308 principes en fonction des critères individuels définis au sein du cadre conceptuel et des autres éléments requis par cette phase. La figure 9 présente un résumé de la méthode appliquée pour cette phase présentée au chapitre 4, en précisant les intrants, les processus sur ces intrants et les extrants de cette phase.

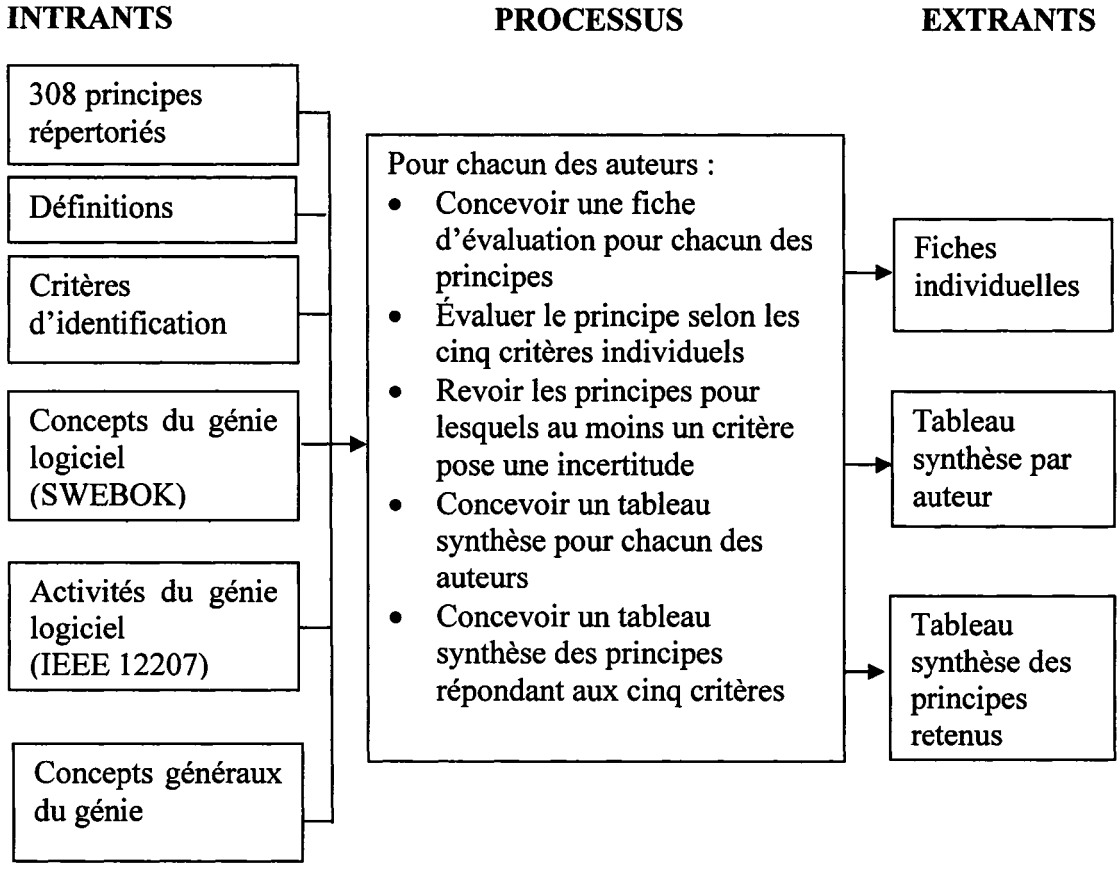

Figure 9 Méthode de recherche pour l'évaluation des principes - phase 2

produira des fiches individuelles[15] pour chacun des principes analysés afin d'assurer l'objectif de la traçabilité énoncé précédemment.

2.2.3 Phase 3 - évaluation selon les critères d'ensemble, catégorisation et liens avec la norme ISO/IEC12207

Cette phase a pour objectif de filtrer encore un peu plus le sous-ensemble des principes issu de la phase 2. Les principes seront catégorisés selon l'une des trois catégories retenues et par la suite, les critères d'ensemble seront appliqués. La figure 10 présente un résumé de la méthode de recherche suivie à cette phase.

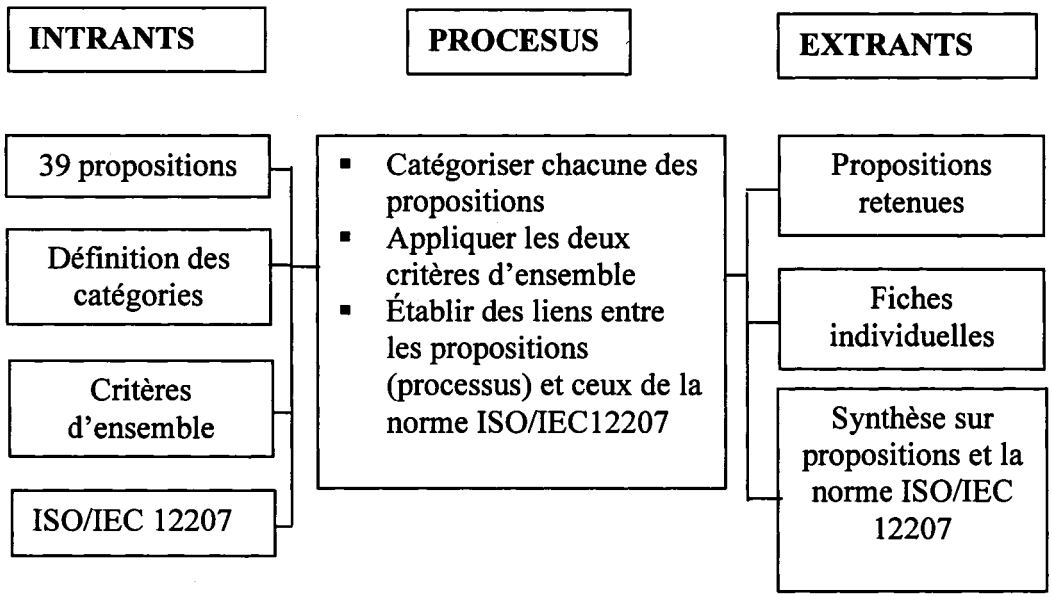

Figure 10 Méthode de recherche pour la phase 3

La phase 3 proposera également une association de chacun des principes retenus appartenant à la catégorie processus avec les processus définis par la norme ISO/IEC

[15] Les fiches individuelles se retrouvent en annexe.

12207. Cette association sera utile à la phase 4 qui portera sur la vérification du degré de couverture des principes avec le corpus des normes du génie logiciel de l'IEEE.

À la fin de la phase 3, nous aurons une liste réduite de propositions qui satisferont l'ensemble des critères individuels et d'ensemble.

2.2.4 Phase 4 - évaluation du degré de couverture des principes retenus

La liste des propositions issue de la phase 3 devra être vérifiée afin d'évaluer le degré de couverture des propositions en fonction de deux vues distinctes. La première vue est le modèle d'ingénierie présenté par Moore (2006).

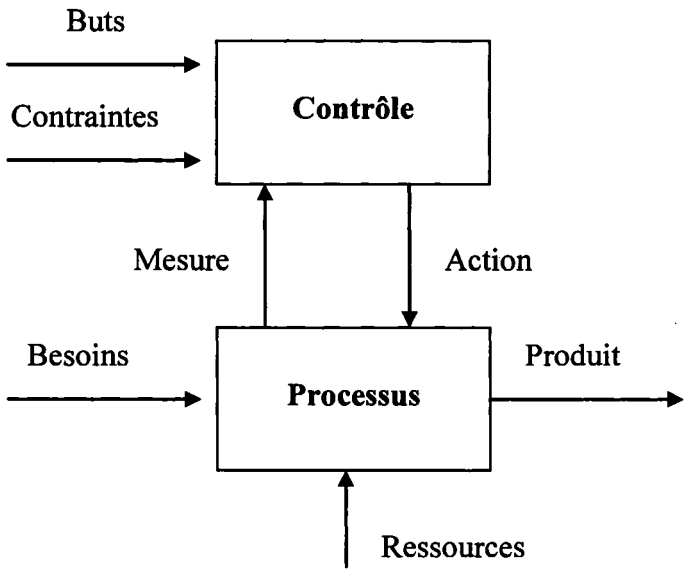

Figure 11 Modèle d'ingénierie Moore (2006)

Chacune des propositions retenue sera analysée en fonction des éléments du modèle d'ingénierie. L'objectif est de vérifier si l'ensemble des éléments du modèle d'ingénierie de Moore est couvert par les propositions.

La deuxième vue de la vérification portera sur la couverture des propositions avec le corpus des normes du génie logiciel présenté par l'IEEE. Principalement, l'association des propositions aux normes se fera par l'entremise de la norme parapluie ISO/IEC 12207. À la phase 3, les propositions de catégorie processus seront associées aux processus de la norme ISO/IEC 12207. De son côté, Moore (2006) a associé les processus de cette norme aux différentes autres normes du corpus de l'IEEE. En combinant les deux résultats, il sera possible d'associer les propositions aux normes de l'IEEE et d'en évaluer le degré de couverture.

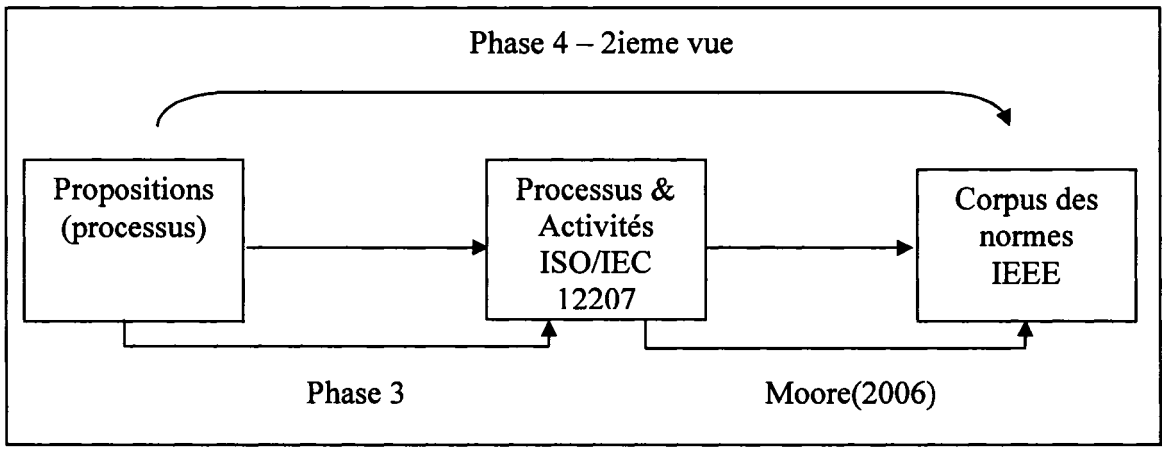

Figure 12 Association des propositions au corpus via la norme ISO/IEC 12207

La méthodologie de recherche proposée se découpe donc en quatre phases principales, tel que présenté à la figure 13.

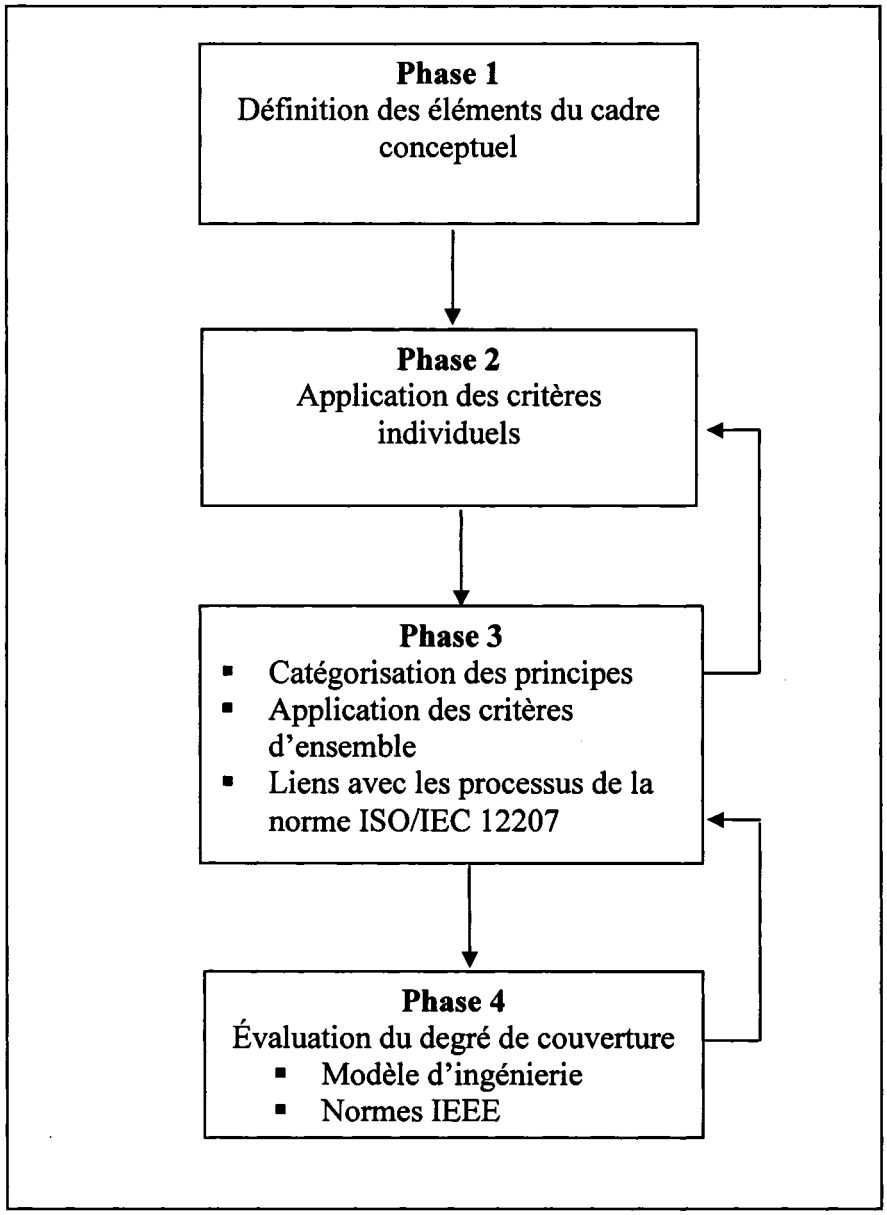

Figure 13 Phases de la méthodologie de recherche

Malgré l'apparence d'une méthodologie de recherche en cascades, il y a des aspects itératifs dans la démarche. Principalement, à l'intérieur de chacune des phases, le travail se déroulera d'une façon itérative. Ainsi, aux phases 2 et 3, les propositions qui poseront des interrogations seront mises de côté et réévaluées de nouveau par la suite. De plus, à

une phase subséquente, il est possible de revenir à une phase antérieure pour apporter des précisions. Comme la méthodologie de recherche raffine de plus en plus l'évaluation des propositions, des retours sont possibles aux phases antérieures afin d'effectuer certains ajustements pour conserver la cohérence globale.

Enfin, une synthèse des résultats sera présentée au chapitre 7. La synthèse fera le bilan des travaux faits par rapport aux travaux antérieurs et dégagera les bénéfices de cette thèse analytique. De plus, les axes de recherches en cours et à venir seront aussi présentés.

CHAPITRE 3

PHASE 1 - DÉFINITION DU CADRE CONCEPTUEL

La phase 1 de la recherche consiste à définir les principaux éléments du cadre conceptuel présenté au niveau de la description de la méthodologie de recherche. Les éléments en extrant de la phase 1 seront :

- Les concepts à la base du génie
- Les concepts du génie logiciel
- Les concepts de l'informatique comme principale discipline frontière
- Les activités du génie logiciel présentées par la norme ISO/IEC 12207
- Les définitions des termes concept et principe

3.1 Les concepts à la base du génie

Bourque et al. (2002) soulignent que les principes fondamentaux du génie logiciel devraient être un sous-ensemble spécifique de principes plus généraux du génie. De même, Moore (1998) souligne que: *« The principles of SE would be regarded as selected, adapted and specialized from principles of engineering in general » (p.5).*

Une avenue logique serait d'identifier les principes fondamentaux du génie, ce qui permettrait de baliser la démarche pour notre recherche. Cependant, on constate que peu d'auteurs ont écrit sur le génie en général et encore moins sur l'identification des principes fondamentaux du génie. La forte spécialisation des disciplines du génie pourrait être une des sources de cette carence.

Dans le but d'apporter certaines balises à notre cadre conceptuel de recherche, la présente section a pour objectif de définir le génie et son essence, de le situer par rapport à la science, à l'art et d'identifier les principaux concepts qui le caractérisent.

Ces concepts seront utiles pour la suite de cette recherche afin de pallier au manque de principes identifiés pour le génie. Pour atteindre ces objectifs, une synthèse a été faite des propos et de la vision de trois auteurs ayant contribués à décrire ce qu'est le génie et le travail de l'ingénieur (Aslaksen (1996), Davis (1998) et Rodgers (1983)).

3.1.1 Qu'est ce que le génie?

Rodgers (1983) définit le génie comme suit :

> *« Engineering refers to the practice of organising the design and the construction of any artifice which transform the physical world around us to meet some recognised need. » (p.51).*

Rodgers souligne que le terme « engineering » a la forme d'un verbe qui signifie une forme d'action. Cette action se concrétise dans le déroulement des activités de design et de construction d'un artéfact (un produit ou un service) en tenant compte de contraintes de nature économique, sociale et environnementale. Cette action requiert aussi la prise de multiples décisions et de compromis tout au long du processus d'ingénierie. Également, Rodgers affirme que *l'essence* même du génie est la *conception et la production d'artéfact.*

Davis (1998) cite la définition de l'ingénierie donnée par le National Research Council (NRC) :

> *« Business, government, academic, or individual efforts in which knowledge of mathematics and/or natural science is employed in research, development, design, manufacturing, system engineering, or technical operations with the objective of creating and/or delivering systems, products, process, and/or services of a technical nature and content intended for use » (p.33).*

Davis (1998) critique cette définition par le fait qu'elle contient une référence circulaire en comportant en elle-même le terme « engineering ». Cependant, il en extrait trois éléments qui caractérisent le génie :

1. Les mathématiques et les sciences naturelles sont au centre des activités du génie.
2. Le génie met l'accent sur la production d'objets plutôt que sur l'élaboration de théories.
3. Le génie ne vise pas à expliquer le monde, mais à le modifier.

Davis (1998) ajoute que le génie est principalement une façon d'organiser et de gérer les activités de design, de développement et de fabrication en s'assurant que tout sera fait dans les temps prévus, en respectant les budgets et à la satisfaction du client. Durant le déroulement du processus de génie, l'ingénieur doit tenir compte de considérations économiques, de sécurité, de fiabilité et de durabilité. Le processus d'ingénierie comporte un élément important de management des activités techniques.

Davis (1998) souligne que l'ingénieur sait organiser le travail technique des différentes activités du processus. De plus, l'ingénieur est en mesure de donner les directives requises pour la réalisation des activités et d'assurer la responsabilité du projet. Enfin, il doit vérifier les résultats obtenus et respecter le code d'éthique de la profession.

Aslaksen (1996) souligne que le génie a une signification différente selon les personnes. Le génie englobe un large spectre de connaissances et d'activités telles le design, la fabrication, la construction, l'entretien, la recherche et développement et le management. Aslaksen définit le génie à son origine comme étant *« The art of getting things done » (p.14).* Le génie serait une discipline basée sur la science et elle comporterait un élément essentiel de créativité. Aslaksen met l'accent sur le fait que le génie est avant tout un processus multidisciplinaire dont l'objectif est de combler un besoin exprimé par un produit ou par un service considéré comme satisfaisant dans le respect des contraintes

économiques, sociales, environnementales et légales. À cause de ces contraintes omniprésentes tout au long du processus, le génie produit des solutions issues de nombreux compromis. Aslaksen affirme que *l'essence* même du génie est un produit issu du cerveau humain qu'aucune machine ne peut élaborer à cause de la complexité des concepts impliqués.

Aslaksen propose que le génie repose sur deux composants majeurs tel qu'illustré à la figure 14.

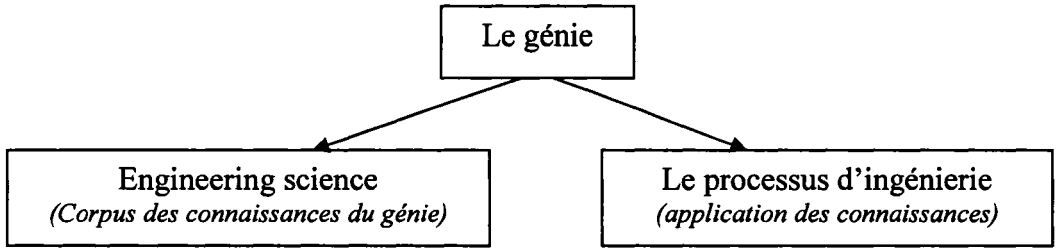

Figure 14 Composant à la base du génie (adaptation des propos de Aslaksen 1996)

Le « engineering science » est définit comme une technologie dont les bases proviennent de la physique et de la chimie (Aslaksen 1996). Rodgers (1983) ajoute la biologie et les mathématiques comme éléments de base du « engineering science ». En plus des connaissances propres au « engineering science », le génie nécessite des connaissances provenant de disciplines frontières telles l'économie, la finance, le marketing et la gestion des ressources humaines (Aslaksen 1996).

Aslaksen (1996) souligne que le génie est fondamentalement un *processus* qui, à partir d'un besoin exprimé, conçoit un produit pour le satisfaire. Ce processus multidisciplinaire joue un rôle intégrateur en intégrant tous les processus individuels en un processus unique et cohérent. Le processus d'ingénierie intègre les activités de développement, de design, de production et la gestion de ces activités. Aslaksen souligne

aussi que ce processus doit être conçu, décrit, paramétrisé et optimisé, et ce même s'il comporte une dimension de créativité. De plus, pour le bon déroulement du processus d'ingénierie, des connaissances spécialisées sont requises, ainsi que des méthodes, des techniques et des outils.

Aslaksen (1996) mentionne que le processus classique d'ingénierie débute après la définition des besoins et se termine par une évaluation du produit issu du processus en fonction des besoins d'origine, de ses performances et de son coût. Même si les besoins de la société sont exprimés à l'extérieur du processus d'ingénierie, l'auteur souligne l'importance que le processus tienne compte de la phase de l'expression des besoins et de son management. L'intégration de la phase des besoins au processus d'ingénierie fait partie des changements nécessaires dans l'évolution du génie.

3.1.2 Caractéristiques du processus d'ingénierie

Aslaksen (1996) souligne que le processus d'ingénierie ne peut se faire d'une façon isolée. Ce processus serait plutôt un composant central interagissant avec d'autres entités telles la recherche, le marketing, la finance et les services d'après vente. Le processus intègre aussi les étapes classiques de la résolution de problème : définition du problème, définition des critères de décisions, identification des solutions possibles et le choix d'une solution. De plus, Aslaksen affirme que la « tracabilité » serait une propriété importante du processus d'ingénierie et que sans celle-ci, le processus perdrait la qualité d'être rationnel. Le processus d'ingénierie serait aussi caractérisé par l'utilisation de méthodes, d'outils et de procédures dont l'efficacité peut être évaluée par la mesure.

Aslaksen (1996) présente trois propriétés fondamentales qui caractérisent le processus d'ingénierie : la complexité, l'incertitude et le risque.

L'auteur affirme que la *complexité* des tâches associées aux projets de génie a décuplé au fil des ans. Une des sources de croissance de la complexité identifiée par l'auteur serait la croissance importante des interactions entre les technologies impliquées dans les projets. De plus, les ingénieurs doivent tenir compte, dès la phase du design, des aspects reliés à l'entretien du produit. Ceci a pour conséquence d'accroître la complexité de la phase du design. Ainsi, le génie ne peut plus se contenter de concevoir un artéfact, mais il doit aussi prévoir l'entretien de cet artéfact et de sa disposition à la fin de sa vie utile.

Aslaksen (1996) affirme que le cerveau humain a des limites importantes pour faire face à la complexité croissante des projets. Ainsi, le cerveau aurait tendance à laisser de côté les détails pour se concentrer sur les principaux éléments. De ce fait, le processus d'ingénierie doit aider les ingénieurs à maîtriser la complexité en les aidant à formuler les différentes tâches à exécuter dans une séquence déterminée afin que le cerveau ne soit pas surchargé. L'auteur cite, à titre d'exemple, la méthode du partitionnement, bien connue dans l'ingénierie des systèmes. Cette méthode permet de diviser chacune des tâches complexes en plusieurs plus petites tâches dont chacune sera plus facilement traitée par le cerveau humain.

Étroitement liée à la complexité, *l'incertitude* serait une autre caractéristique du processus d'ingénierie. À partir du moment où le design doit anticiper les activités d'entretien et que la durée des projets s'étale sur plusieurs années, il est incontournable que des approximations de certaines variables soient faites. De plus, la taille des projets rend difficile le traitement en profondeur de toutes ces variables. Ainsi, le processus d'ingénierie doit composer avec des approximations et des moyennes statistiques. Dans certains projets, les données disponibles sont extraites de modèles à échelle réduite qui n'offrent pas l'exactitude du projet réel. Pour pallier à ces approximations, le génie s'est doté de marges de sécurité qui permettent d'absorber certaines variations entre les approximations et les données réelles (Rodgers (1983)). Aslaksen (1996) mentionne

aussi que le développement du « *statistical engineering* » pourrait fournir des outils afin de mieux affronter la complexité et l'incertitude des projets de génie.

Le *risque* est la troisième caractéristique fondamentale du processus d'ingénierie identifiée par Aslaksen. Le risque peut se manifester par des changements importants à tenir compte en cours de projet et ayant des impacts significatifs sur celui-ci. Aslaksen souligne que le risque est une propriété composite qui se définit comme la probabilité qu'un événement survienne avec ses effets négatifs sur le projet. Le processus d'ingénierie tente de minimiser les deux composants du risque soient la probabilité qu'un événement se manifeste et les impacts négatifs de l'événement sur le projet. Le processus d'ingénierie comporte un volet de gestion du risque. Ce volet tente d'identifier et de contrôler les événements ayant un potentiel d'effets négatifs sur le projet.

3.1.3 Le design : phase prédominante du processus d'ingénierie

Rodgers (1983) souligne qu'une des phases majeures dans le processus d'ingénierie est la phase du design. Le design comporte deux volets : le design industriel et le design d'ingénierie.

Le *design industriel* aborde les aspects cosmétiques et ergonomiques du produit. Les interactions personne-machine sont évaluées en considérant, entre autres, les aspects de sécurité et de santé.

Le *design d'ingénierie* est le volet majeur de la phase du design. Ce volet se concentre sur la conception d'un artéfact qui va répondre au besoin exprimé tout en étant efficace à un coût raisonnable. Il est à noter que l'objectif de la phase du design se rapproche de près de la définition donnée au génie.

Rodgers (1983) souligne que le design est une activité intellectuelle qui procède à l'élaboration et à l'évaluation des solutions possibles en tenant compte des contraintes d'ordre économique, légal et environnemental. Également, le jugement et l'expérience de l'ingénieur sont mis à contribution dans le design. Le design nécessite une série de décisions prises par l'ingénieur et des compromis doivent être souvent faits. L'ingénieur puisera des informations nécessaires à son évaluation dans les synthèses du corpus de connaissances du « engineering science ». Cependant, si l'ingénieur ne trouve pas ce dont il a besoin pour réaliser son design, il peut alors faire appel à des groupes d'ingénieurs-chercheurs du « engineering science » pour développer, par exemple, un nouveau matériau comportant les caractéristiques et les comportements recherchés. Ainsi, des activités de recherche sont possibles et nécessaires pour soutenir certains projets du génie.

Rodgers (1983) souligne que l'ingénieur peut choisir trois orientations possibles pour réaliser le design.

- Faire un design en utilisant les codes de pratiques éprouvés
- Modifier une solution existante et éprouvée, et l'adapter à un nouveau contexte ou à une échelle plus grande
- Remplacer une façon de faire éprouvée par un nouveau concept qui rendra désuète les solutions existantes

Vincenti (1990) regroupe les deux premières orientations sous le titre du *normal design*, alors que la troisième orientation est nommée *radical design*. À cause des aspects de sécurité, légaux et des conséquences catastrophiques d'un échec, le design sera imprégné d'un esprit favorisant l'évolution de solutions (design normal), plutôt que la recherche d'une solution révolutionnaire (design radical).

Au niveau du design, l'ingénieur doit tenir compte de plusieurs facteurs, parfois même contradictoires. De plus, il doit tenir compte des conseils et des recommandations de plusieurs spécialistes tant au niveau technologique qu'au niveau économique, environnemental et social. Ainsi, le processus d'ingénierie serait ponctué d'une suite de compromis entre l'idéal à atteindre et les diverses contraintes dont l'ingénieur doit tenir compte.

3.1.4 Le génie et la science

Rodgers (1983) souligne que la science trouve ses racines dans une discipline nommée la philosophie naturelle. Cette discipline était basée essentiellement sur l'observation des phénomènes naturels sur lesquels les philosophes de l'époque émettaient des hypothèses sur leurs structures et leurs comportements. La philosophie naturelle a évolué vers la science telle que connue aujourd'hui lorsque le vocabulaire du langage écrit a atteint un niveau de maturité suffisant pour identifier, expliquer et représenter les phénomènes expliqués. À l'instar des scientifiques, les philosophes naturels recherchaient des explications pouvant être vérifiées par des expérimentations répétables et donnant des résultats stables. Ces expériences répétables étaient un pré requis à un consensus entre les philosophes.

Rodgers (1983) souligne que le scientifique a comme motivation d'expliquer les phénomènes et les choses du monde réel. Également, les scientifiques auraient tendance à rechercher des théories qui provoquent une révolution dans les concepts d'un domaine pointu et être ainsi des aspirants aux prix Nobel. Cependant, ces propos de Rodgers sont fortement nuancés par Thomas Kuhn (1970) à l'effet que la majorité des scientifiques exécutent leurs travaux dans un contexte de science dite normale. Dans ce contexte, les scientifiques basent leurs travaux sur des contributions scientifiques antérieures associées à un paradigme spécifique. Un paradigme définit les éléments de base, comme les théories, les lois, les méthodes, les instruments et les orientations de solutions aux

problèmes reconnus par le paradigme. Associé de près à un paradigme, une communauté scientifique est formée et partage, sous forme de consensus, les éléments du paradigme scientifique. Cette communauté fera l'évaluation des travaux de recherche effectués au sein du paradigme.

Ainsi, à l'encontre des propos de Rodgers, Kuhn affirme que la majorité des travaux de recherche scientifique n'a pas pour objectif de produire des révolutions scientifiques. Le but de la science normale est d'améliorer, de raffiner, de préciser et de valider les éléments de base du paradigme. Kuhn souligne que la révolution scientifique n'est pas exclue, mais qu'elle ne se présente que très rarement. Une révolution scientifique bouleverse les paradigmes en place soit en les modifiant en profondeur ou en les remplaçant tout simplement.

La science est basée sur un système déductif raffiné qui ne laisse pas de place à l'approximation qui pourrait fausser le choix d'une théorie par rapport à une autre. La science peut être précise. Les expérimentations sont réalisées à l'aide d'instruments précis, le nombre de variables est limité et l'environnement est rigoureusement contrôlé. Historiquement, les résultats des expérimentations de la science ont pu être généralisés sous forme de lois et reléguer aux oubliettes plusieurs croyances de l'époque à l'effet que les nombreux *dieux* étaient responsables de tous les événements du monde naturel. Le scientifique fait aussi preuve de créativité dans son travail. Cette créativité est motivée par l'explication des phénomènes du monde naturel et le sentiment de fierté d'effectuer des découvertes.

Le génie trouve ses racines au niveau de la construction de « crafts » tels des outils et des machines. Ces « crafts » permettaient de faciliter le travail des gens de l'époque en réalisant des tâches difficilement faisables à la main. La conception de « craft » trouve son origine bien avant la philosophie naturelle et la science, puisque le « craft » ne

requérait que de la dextérité et de la créativité manuelle. Quant à la science, elle requiert un langage écrit évolué et des éléments des mathématiques.

À l'instar des scientifiques, les ingénieurs font eux aussi des expérimentations. Cependant, celles-ci se concentrent sur les caractéristiques macroscopiques des objets comme le comportement des matériaux dans un contexte donné avant de les utiliser dans un projet. De ces expérimentations, les connaissances acquises sont consolidées dans le « handbook » de la spécialité sous forme de théories, de règles ou de formules. L'ingénieur fait aussi preuve de créativité dans sa passion de concevoir des objets utiles et surtout dans ceux que l'on ne pensait pas réalisables.

Rodgers (1983) souligne que le corpus de connaissances du génie diffère de celui de la science sous trois aspects. En premier lieu, les buts et objectifs de chacun diffèrent. La science tente de comprendre le monde qui nous entoure alors que le génie tente de combler les besoins de la société en modifiant le monde qui nous entoure.

En deuxième lieu, la science et le génie reposent sur des présuppositions différentes. La science présuppose une certaine régularité dans la nature alors que le génie présuppose que la nature peut être modifiée et manipulée afin de satisfaire les besoins de la société. Le progrès au niveau de la science est axé sur de meilleures théories, de meilleurs concepts permettant de mieux expliquer et prédire les phénomènes. La généralisation des théories sous forme de lois est aussi un autre volet du progrès scientifique. Au niveau du génie, le progrès peut s'observer sous deux volets : le produit et le processus de fabrication du produit. Ainsi, les produits créés peuvent être plus performants ou rendre caduques d'autres produits. Au niveau du processus, le progrès permet de fabriquer les produits plus efficacement du point de vue économique et environnemental.

En dernier lieu, le génie est plus préoccupé que la science par les considérations sociales, économiques et environnementales. Le génie tient aussi compte des aspects de sécurité, de fiabilité, de durabilité et d'esthétique. Le génie est aussi une activité beaucoup plus visible tant au niveau des succès que des échecs. La science est une activité plutôt privée en laboratoire où les préoccupations sociales et économiques ne seraient pas au centre des préoccupations des scientifiques. Cependant, la science n'est pas exempte de ces préoccupations. À titre d'exemple, les manipulations génétiques et leurs conséquences sur l'humanité sont débattues par la communauté scientifique.

Rodgers (1983) propose une classification des connaissances techniques du génie présentée à la figure 15. L'ingénieur doit recourir à plusieurs technologies pour atteindre ses objectifs. Ainsi, ces technologies se regroupent sous deux catégories : les *matériaux* et les *produits*. Afin de soutenir ces catégories, le génie dispose aussi d'une couche de connaissances plus théorique nommée le « *engineering science* ». Le « engineering science » représente la science de nature analytique à la base du génie. Cette catégorie de connaissances effectue le lien entre le génie et la science. Le « engineering science » partage des caractéristiques communes avec la science. Les connaissances se développent à partir des problèmes rencontrés et des solutions trouvées. Ainsi, les connaissances acquises sont cumulatives et enrichissent continuellement le corpus des connaissances propres au génie. De ce fait, Rodgers (1983) affirme que la caractéristique du « *selft generating* » des connaissances de la science s'applique aussi au génie par l'entremise du « engineering science ». Le génie a des problématiques spécifiques qui stimulent les chercheurs du génie à explorer, expérimenter et à trouver des explications et des solutions.

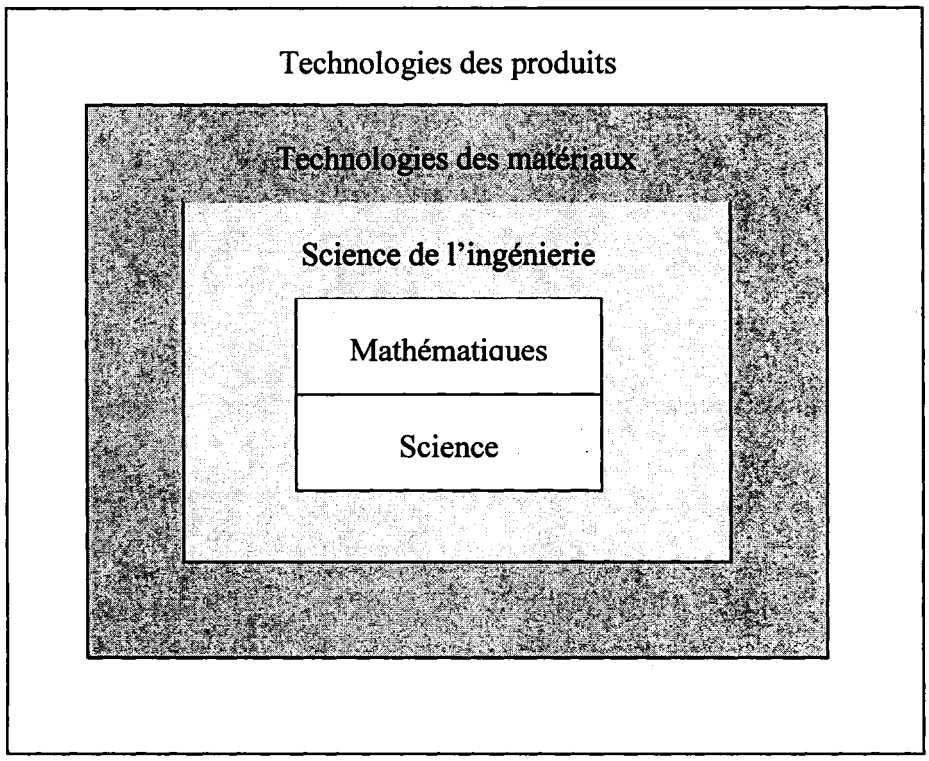

Figure 15 Classification des connaissances techniques du génie (adaptation du modèle de Rodgers (1983))

Les théories empiriques obtenues aident à analyser de nouvelles situations. Le « engineering science » se développe sur une base continue et alimente les catégories *produits* et *matériaux* du génie.

Le « engineering science » se distingue de la science par les caractéristiques suivantes. En premier lieu, les théories du génie peuvent très bien soutenir le design et être très utiles en pratique, mais ne pas être adéquates du point de vue scientifique. Le succès en pratique d'une théorie ne la valide pas automatiquement. Le génie doit aussi composer avec des approximations qui ne pourraient pas être acceptables du point de vue de la science. En dernier lieu, le génie ne peut se permettre de falsifier ses résultats compte tenu de la responsabilité professionnelle et des conséquences importantes des échecs généralement bien médiatisés.

3.1.5 Le génie et les arts

Le génie comporte un volet essentiel de créativité. Cette créativité amène à se demander si le génie a des liens avec l'art.

Selon Rodgers (1983), l'art concerne essentiellement l'exploration et l'expression des émotions conscientes ou non d'une personne (l'artiste) par des moyens tels la littérature, les arts graphiques et la musique. Ainsi, l'art est une activité de nature émotive tandis le génie est une activité plutôt rationnelle. La créativité en art est motivée par l'expression des émotions de l'artiste face à des situations du monde externe.

Jusqu'au 18^{ième} siècle, le mot art référait plutôt à un « craft » ou à de l'expertise spécialisée telle, entre autres, un sculpteur, un menuisier et un médecin. Par la suite, une distinction s'est faite entre les beaux arts et l'art utile. Également, une distinction s'est aussi faite entre l'artiste et l'artisan.

Rodgers (1983) souligne que le génie se distingue des arts selon trois caractéristiques. En premier lieu, le génie est une *activité planifiée* et le *résultat à obtenir est déterminé* avant le début du processus d'ingénierie. La finalité en art n'est pas toujours planifiée. L'artiste peut faire évoluer son œuvre au fil de l'exploration de ses émotions durant la réalisation de son œuvre.

En second lieu, au niveau du génie, il est possible de distinguer les matériaux de base du produit final. À son tour, le produit final peut servir de matériel de base à un autre produit. Il est ainsi possible d'établir une classification hiérarchique (i.e. est composé de) des différents produits et matériaux. En arts, cette distinction est plus futile et il est souvent difficile de distinguer les deux. De plus, une œuvre peut rarement servir de matériel de base à une autre œuvre. La classification hiérarchique en arts est donc difficilement réalisable.

En dernier lieu, le génie peut différencier les moyens utilisés pour fabriquer le produit, du produit lui-même. En arts, cette distinction n'est pas aussi claire. Un poète n'a pas besoin d'outils pour composer son œuvre. L'utilisation de mots et d'un crayon ne serait pas considérée comme des moyens au même titre que ceux utilisés par le génie. Une autre personne n'arriverait pas au même résultat, même en connaissant les mots et en utilisant le même crayon.

Le génie et l'art partagent une caractéristique commune. Ainsi, les deux disciplines se caractérisent par la création d'artéfacts et non de théories pour les expliquer. La créativité n'est pas réservée uniquement aux arts puisqu'on la retrouve aussi en science et en génie.

Le génie ne serait donc ni de l'art, ni de la science pure, mais il se situerait quelque part entre les deux disciplines. De plus, l'utilisation de l'expression en génie de l'état de l'art ne serait pas appropriée. Rodgers souligne à cet effet que l'utilisation de l'état de la technologie serait plus appropriée, tout en reconnaissant que la formule est moins élégante. On peut aussi ajouter que l'expression de l'état de l'art fait souvent référence à une synthèse sur un sujet spécifique.

3.1.6 L'éducation en génie

Le génie trouve ses racines dans les activités militaires. En 1676, en France, on assiste à la création du premier groupe d'ingénieurs nommé « le corps du génie » (Davis 1998). Les membres de ce groupe étaient exclusivement des officiers militaires spécialisés en artillerie et en travaux de génie militaire tel des ponts d'appoint. Le « corps du génie » n'était pas une école, mais un regroupement de spécialistes. Ce groupe était considéré comme le « gardien » du savoir-faire militaire entre les guerres.

En 1690, la France crée la première école d'artillerie afin de répondre au besoin d'offrir une formation structurée aux ingénieurs de cette spécialité (Rodgers 1983). Toujours en France, en 1716, le « corps des ponts et chaussées » voit le jour. L'objectif de ce groupe était de bâtir et de maintenir de réseaux des routes et des canaux navigables de la France.

Les français ont été ainsi les premiers à reconnaître la nécessité d'offrir une formation structurée aux ingénieurs. En 1749, « l'École du génie » fut créée. Le corps professoral de cette école entreprit d'écrire une synthèse des connaissances du génie. Les thèmes traités portaient sur les structures, les matériaux, l'hydraulique, la mécanique et les mathématiques appliquées. L'aboutissement de ces travaux mènera à la création de l'École polytechnique en 1794. L'École offrait dès ses débuts plusieurs spécialisations du génie. De plus, elle offrait un cheminement rigide composé d'aspects théoriques et pratiques d'une durée de quatre ans. Les trois premières années composaient le tronc commun de tous les étudiants et la quatrième année était celle de la spécialisation. L'École appliquait une sélection stricte au niveau de l'admission des candidats. Le modèle de l'École polytechnique servit de modèle à la création de plusieurs autres écoles de génie dans le monde telle l'Académie militaire de West Point aux USA en 1817. Cette dernière est considérée comme la première école de génie aux États-Unis (Davis 1998).

Vers 1850, on assiste à la mise en place du modèle moderne d'éducation en génie. Ce modèle comporte un *curriculum formel*, un *examen de certification* à la fin des études et la constitution d'un *code d'éthique* régissant la profession.

Davis (1998) souligne que l'origine militaire du génie a teinté la profession. D'une part, les ingénieurs militaires de l'époque devaient accorder une attention particulière à la fiabilité des objets conçus (les canons, les bombes, les fusils, etc). Ainsi, les ingénieurs procédaient à des *tests systématiques* des matériaux et des procédures de fabrication avant la production en volume. D'autre part, ces ingénieurs étaient avant tout des

officiers de l'armée formés d'une façon *disciplinée* et aptes à prendre en charge des responsabilités. Également, la formation en mathématiques et en physique permettait à ces ingénieurs d'aborder les problèmes d'une façon *systématique.*

Le titre d'ingénieur est contrôlé par les corporations ou les ordres d'ingénieurs. Ainsi, dans certains pays, l'ingénieur doit être reconnu et obtenir une licence de l'ordre des ingénieurs pour exercer sa profession. Les corporations d'ingénieurs peuvent poursuivre les faux ingénieurs et suspendre le permis de pratique d'un ingénieur qui n'aurait pas respecté le code d'éthique ou qui serait reconnu coupable de fautes professionnelles.

3.1.7 Concepts du génie

Dans les sections antérieures, il a été souligné que le génie s'appuie sur deux assises constituant sa fondation. La première est le corpus de connaissances du génie représenté par le « engineering science ». La seconde est le processus d'ingénierie qui applique les connaissances du « engineering science » dans la conception d'artéfact. Le produit ou service issu du processus d'ingénierie doit satisfaire les besoins et les exigences exprimés par le client. Le processus sera réalisé par des individus (les ingénieurs) impliqués dans les activités. La figure 16 présente les concepts généraux du génie.

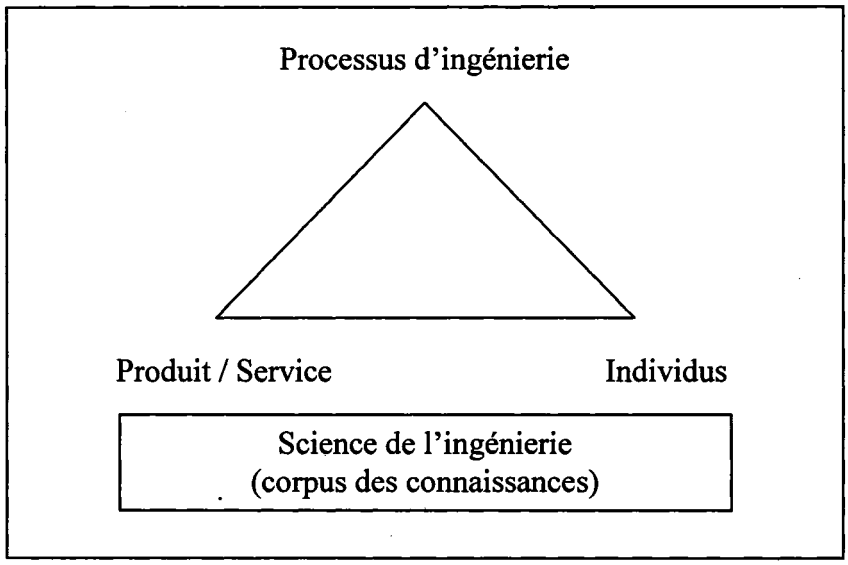

Figure 16 Concepts généraux du génie

3.1.7.1 Processus d'ingénierie

Aslaksen (1996) souligne que le processus d'ingénierie est un composant majeur du génie. C'est au sein de ce processus que les connaissances du génie sont appliquées. Le processus d'ingénierie est composé d'une suite d'activités intellectuelles et rationnelles qui débute avec la phase de définition des besoins. L'objectif du processus est de produire l'artéfact (produit/ service) qui répondra aux besoins du client et qui tient compte des contraintes imposées. La figure 17 présente une modélisation des activités principales du processus d'ingénierie.

Définition des besoins	Design	Construction Fabrication	Entretien
Management des activités			
Activités de recherche et développement			

Figure 17 Activités du processus d'ingénierie (modélisation du texte de Aslaksen 1996)

Le processus d'ingénierie doit être planifié, décrit, évalué et optimisé (Aslaksen 1996). De plus le processus doit permettre la traçabilité des choix et des décisions prises tout au long de la réalisation des activités. Le processus intègre d'une façon cohérente d'autres processus, des méthodes, des techniques, des outils et des normes pour standardiser le déroulement du processus, tel que présenté à la figure 18. Également, le processus dans son entier fait l'objet d'une planification en divers volets tels, les plans de projet, de risques, etc. Le processus d'ingénierie a comme caractéristiques d'être complexe, de comporter des risques et de l'incertitude (Aslaksen 1996). Cependant, ces caractéristiques ne sont pas exclusives à l'ingénierie.

Plans (planification)	N
Processus	o
Méthodes	r
Techniques	m
Outils	e
	s

Figure 18 Principaux concepts associés au processus d'ingénierie

3.1.7.2 Produit ou service

Le produit ou le service conçu est l'aboutissement du processus d'ingénierie. Le produit est le résultat des décisions et des compromis faits tout au long du processus depuis la définition des besoins jusqu'à sa fabrication et tests. À l'origine du génie, lorsque les ingénieurs étaient avant tout des officiers militaires, les armes ou produits militaires conçus se devaient d'être fiables. Des bombes qui n'explosent pas, des fusils qui s'enraillent et autres équipements qui se brisent sont, entre autres, des exemples d'événements qui peuvent changer l'issue d'une bataille. Ainsi, la fiabilité du produit serait dès l'origine du génie, une qualité recherchée par les ingénieurs. Pour y arriver, les ingénieurs-officiers de l'époque ont mis en place des procédures systématiques de tests pour valider la fiabilité du produit. Le produit peut aussi posséder d'autres qualités telles la durabilité, la faciliter d'entretien, la possibilité de la fabriquer économiquement. De plus, le produit doit aussi se conformer aux contraintes environnementales et légales[16].

Le produit doit évidemment satisfaire les besoins exprimés par le client. Ainsi, la définition des besoins est une activité importante qui se doit d'être bien faite dans la mesure où les activités subséquentes s'y réfèrent. Tel que souligné par Aslaksen (1996), cette activité doit être incluse dans le processus d'ingénierie, ainsi que son management.

3.1.7.3 Les individus

Les ingénieurs ont reçu une formation formelle et accréditée par les corporations d'ingénieurs. L'ingénieur a été formé pour aborder les problèmes et les résoudre d'une façon systématique i.e. avec méthode, dans un ordre et un but déterminé. Découlant des origines militaires du génie, l'ingénieur effectue son travail avec un sens du devoir, de l'ordre et de l'obéissance. Les ingénieurs de l'époque étaient en premier lieu des

---

[16] Ces contraintes ne sont pas toujours respectées dans le contexte militaire.

officiers militaires et il était donc naturel qu'ils organisent et dirigent les travaux et qu'ils assument la responsabilité des résultats obtenus. Même si l'ingénieur moderne n'est plus d'office un militaire, il est formé à organiser le travail à effectuer, à prendre des décisions et à assumer la responsabilité de ses actes. L'ingénieur doit aussi se conformer et respecter le code d'éthique de sa profession. Il doit garder à l'esprit qu'il engage sa responsabilité professionnelle dans chaque activité qu'il réalise. L'ingénieur pris en défaut peut être sanctionné par l'ordre des ingénieurs et à l'extrême se voir radier et ainsi ne plus être en mesure d'exercer sa profession.

Le tableau XVII résume les concepts principaux à la base génie identifiés dans ce chapitre. Ces concepts, entre autres, serviront de référence dans l'évaluation des principes du génie logiciel. Ils seront utilisés pour établir des critères permettant d'identifier des principes se rattachant au génie.

Tableau XVII

Concepts principaux à la base du génie

Processus d'ingénierie	**Produit ou service**	**Individus**
Processus doit être : ▪ Planifié ▪ Décrit ▪ Évalué ▪ Optimisé Intègre les activités : ▪ Définition des besoins ▪ Design ▪ Fabrication ▪ Entretien ▪ Management ▪ R & D	Satisfaisant les besoins exprimés par le client Qualités ▪ Fiable ▪ Durable Respectant les contraintes ▪ Économiques ▪ Environnementales ▪ Légales ▪ Sociales ▪ Des matériaux	À reçu une formation formelle et accréditée Apte à appliquer des connaissances techniques et théoriques Reconnu comme ingénieur par l'Ordre Respecter le code d'éthique Être en mesure de prendre des responsabilités

Tableau XVII (suite)

Processus d'ingénierie	**Produit ou service**	**Individus**
Caractérisé par ▪ La complexité ▪ L'incertitude ▪ Les risques		Engager sa responsabilité professionnelle Aborder les problèmes d'une façon systématique, disciplinée et avec créativité

3.1.8 Modèle du génie

Moore (2006) présente un modèle intéressant schématisant le processus d'ingénierie. Ce modèle modélise le processus d'ingénierie qui à partir des besoins exprimés et des ressources réalisera un produit satisfaisant aux attentes du client. Ce modèle précise également, le volet contrôle du processus. Le contrôle s'alimente en mesures diverses au sein du processus. Ces mesures sont, pas la suite analysées en considérant, entre autres, les buts et les contraintes du projet, afin de corriger le processus via une action. La figure 19 présente le modèle de Moore (2006).

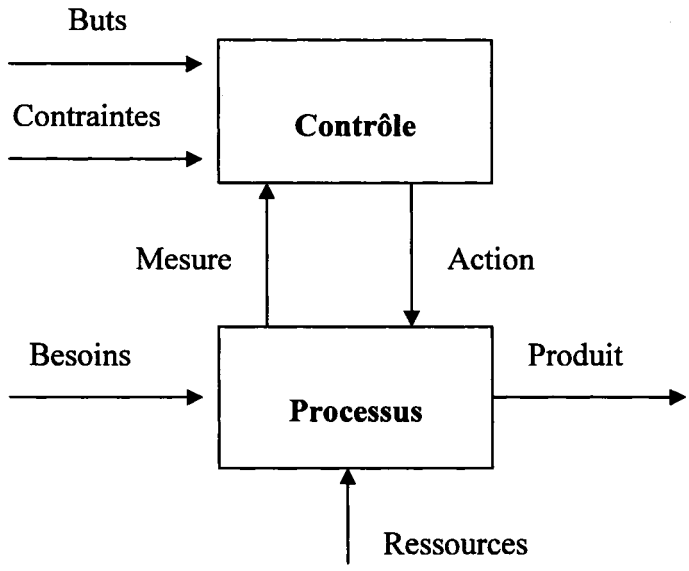

Figure 19 Modèle de l'ingénierie (Traduit de Moore 2006)

3.2 Le génie logiciel

L'expression « software engineering » fut le titre donné à une conférence de l'OTAN en 1968. Comme le rapporte Naur et al. (1968), le choix de ces termes avait pour objectif de sensibiliser les développeurs à l'effet que le logiciel devrait être produit en utilisant des connaissances théoriques et pratiques tel que fait dans les disciplines établies du génie. Cette conférence rassembla des experts en développement de logiciels intéressés à trouver et à explorer des pistes de solutions pour faire face aux problèmes identifiés tels, entre autres, la croissance de la complexité dans le développement des logiciels, les dépassements de coûts et d'échéanciers ainsi que les aspects de fiabilité et d'entretien du logiciel.

Dans le cadre de cette recherche, nous ne traiterons pas du sujet délicat à savoir si le génie logiciel est une discipline du génie à part entière ou s'il est plutôt un sous-domaine de l'informatique. Comme les corporations d'ingénieurs reconnaissent maintenant les programmes de génie logiciel menant au titre d'ingénieur, nous prenons pour acquis que le génie logiciel est une nouvelle discipline du génie en pleine émergence. Cependant, les définitions du génie logiciel abondent, tantôt se rapprochant de l'informatique appliquée et tantôt se rapprochant du génie. Pour le cadre de cette recherche, nous adoptons la définition du génie logiciel tel que cité dans le guide du corpus de connaissances du génie logiciel (SWEBOK 2004). Ce guide a fait l'objet de trois versions majeures, soutenues par des organismes réputés telles IEEE Computer Society et ISO (ISO TR19760). De plus, SWEBOK a été revu par plusieurs centaines d'experts internationaux et son contenu fait l'objet d'un large consensus concernant les connaissances acceptées par la communauté du génie logiciel, ce qui représente un atout significatif.

Le guide SWEBOK a retenu la définition du génie logiciel donnée par la IEEE Computer Society :

> *« The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (p.1-1).*

Cette définition donnée au génie logiciel est en fait presque la même que l'IEEE a donné au terme génie (norme IEEE 610). Ainsi, on y retrouve les concepts importants associés au génie tels l'approche *systématique*, *disciplinée* et *quantifiable*, ainsi que les activités de développement et d'entretien.

Les connaissances référencées par le guide SWEBOK font l'objet d'un consensus à l'effet qu'elles sont généralement utilisées et requises dans un projet de développement d'un logiciel. Le guide se divise en dix domaines de connaissances du génie logiciel tel que présenté à la figure 20. À ces domaines, s'ajoutent un chapitre sur les disciplines frontières du génie logiciel.

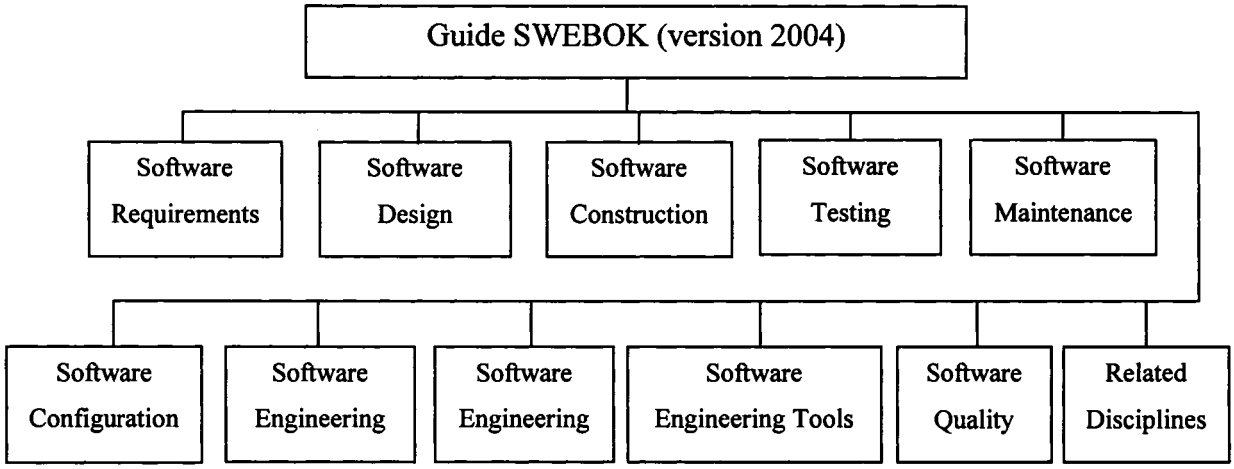

Figure 20 Domaines de connaissances du guide SWEBOK (Abran et al. 2004)

3.2.1 Les concepts du génie logiciel

Nous allons maintenant présenter sommairement chacun des domaines de connaissances et identifier les principaux concepts de chacun. Ces concepts seront nécessaires pour l'analyse des principes qui suivra. Les concepts sont extraits à partir de la structure de base de chacun des chapitres de SWEBOK. Ainsi, un tableau est produit pour chacune des sections d'un domaine de connaissances. La colonne *sous-section* représente les sous-sections. Nous avons conservé le titre exact des sous-sections en anglais pour éviter des confusions de traduction inutiles considérant que les principes à analyser sont tous formulés en anglais. La colonne *concept* représente les concepts présents dans chacune des sous-sections. Seuls les concepts sont énumérés et non un résumé du contenu de la sous-section. Les prochaines sections présentent essentiellement des tableaux, ainsi, le lecteur peut à sa guise passer directement à la section 3.2.2.

3.2.1.1 Les exigences logicielles (Software Requirements)

Ce domaine de connaissance se concentre sur l'élucidation, l'analyse, la spécification et la validation des exigences logicielles. Ces activités sont fondamentales dans le processus de développement de logiciel. Les exigences concrétisent les besoins exprimés par le client et les contraintes spécifiques auxquelles le futur logiciel devra se conformer. Une exigence est une propriété que le logiciel doit comporter afin de répondre aux demandes du client. Le domaine de connaissance est subdivisé en sept sections. Nous présentons sept tableaux mettant en évidence les concepts de chacune des sections. Il est à noter que les concepts ont été laissés en langue originale anglaise afin d'éviter les problèmes de traduction. Compte tenu que l'ensemble des 308 propositions analysées sont toutes en anglais, il est primordial d'éviter les traductions qui peuvent amener des flous.

Tableau XVIII

Concepts de la section "Software Requirements Fundamentals"

Software Requirement Fundamentals	
Sous-sections	Concepts
Definition	Verifiable property which must be exhibited by software to solve a particular problem
Product and Process Requirements	▪ Product: requirement on software ▪ Process: a constraint on the development of software
Functional and Non-functional Requirement	▪ Functions that the software is to execute ▪ Constraints or quality requirement ▪ Performance, maintainability, safety, reliability
Emergent Properties	Requirement which depend on integration of component
Quantifiable Requirements	Can be verified
System Requirements and Software Requirement	▪ System : the whole system ▪ Software : requirement derived from system

Tableau XIX

Concepts de la section "Requirement Process"

Requirements Process	
Sous-sections	Concepts
Process Models	▪ Iterative activity ▪ Requirement part of Configuration Management ▪ Must adapted to the organization and project context ▪ Elicitation, analysis, specification, validation
Process Actors	▪ Users, Customers, Marketing, regulator, engineers
Process Support and Management	▪ Cost, human resources, training, and tools
Process Quality and Improvement	▪ Quality standard, Customer's satisfaction ▪ Software Quality Attributes, Measurement and Benchmarking ▪ Improvement planning and implementation

Tableau XX

Concepts de la section "Requirements Elicitation"

Requirements Elicitation	
Sous-sections	Concepts
Requirement Sources	▪ Business goals, Domain knowledge, Stakeholders ▪ Operational & Organizational environment
Elicitation Techniques	▪ Interviews, Scenarios, Prototypes, Meetings, Observations

Tableau XXI

Concepts de la section "Requirements Analysis"

Requirements Analysis	
Sous-sections	Concepts
	▪ Detect and resolve conflicts between requirements ▪ Bounds of the software ▪ System requirements
Classification	▪ Functional and Non-Functional ▪ Derived from a high level requirement ▪ Product or Process requirement ▪ Priority and Scope ▪ Volatility or Stability of requirement
Conceptual Modeling	▪ Data and Controls flows, State models, Event Traces ▪ User Interaction, object models, data models
Architectural Design and Requirement allocation	▪ Software architect, Software components ▪ Requirement allocation to components
Requirement negotiation	▪ Conflict resolution : o Stakeholder o Between requirement and resources o Between functional and non-functional requirement

Tableau XXII

Concepts de la section "Requirement Specification"

Requirements Specification	
Sous-sections	Concepts
System Definition Document	▪ Concept of operations ▪ High level system requirements from the domain perspective ▪ IEEE 1362
System Requirement definition	▪ System engineering ▪ IEEE 1233
Software Requirement Specification	▪ The basis for agreement between customers and contractors or suppliers ▪ Rigorous assessment of requirements ▪ Basis for estimating product costs, risks and schedules, software enhancement, acceptation tests

Tableau XXIII

Concepts de la section "Requirement validation"

Requirements validation	
Sous-sections	Concepts
Requirements Reviews	▪ Inspection or Reviews Requirements o Errors, mistaken assumptions, lack of clarity and deviation from standard practice o Checklist
Prototyping	▪ Validating the software engineer's interpretation of the requirements ▪ Eliciting new requirements ▪ Feed-back from users
Model validation	▪ Static analysis and Formal reasoning
Acceptance Tests	▪ Validate the final product

Tableau XXIV

Concepts de la section "Practicals considerations"

Practical Considerations	
Sous-sections	Concepts
Iterative nature of the requirement process	▪ Iterate to towards a level of quality and detail ▪ Revision of requirements (changes)
Change Management	▪ Change management procedures ▪ Change analysis
Requirement Attributes	▪ Classification ▪ Acceptance Test plan ▪ Identifier : unique and unambiguous
Requirement tracing	▪ Source of requirement (backwards) ▪ Impact of changes ▪ To design entities (forwards)
Mesures	▪ Volume of requirements (number) ▪ Size of changes (evaluating) ▪ Cost estimation (development & maintenance0

3.2.1.2 Le design du logiciel (Software Design)

Le design est un processus qui fait l'analyse des exigences logicielles dans le but de concevoir une description de l'architecture interne du logiciel et de ses composants. L'étape du design peut produire plusieurs solutions possibles. Celles-ci seront évaluées et une sera choisie. Le design comprend deux activités principales. En premier lieu le design architectural exposera les grandes lignes de la solution proposée et identifiera ses principaux composants. Par la suite, le design détaillé prendra en charge la description de chacun des composants identifiés à un niveau de détail suffisant pour le construire (le programmer). Les concepts du design se regroupent en six sections au niveau des six prochains tableaux.

Tableau XXV

Concepts de la section "Software design fundamentals"

Software Design Fundamentals	
Sous-sections	Concepts
General Design Concepts	▪ Problem-solving ▪ Goals, constraints, alternatives, representations, and solutions.
Context of Software Design	▪ Phase of software engineering life cycle
Design process	▪ Architectural Design ▪ Detailed Design
Enabling techniques	▪ Abstraction ▪ Coupling and Cohesion ▪ Decomposition and modularization ▪ Encapsulation / information hiding ▪ Separation of interface and implementation ▪ Suffiency, completeness and primitiveness

Tableau XXVI

Concepts de la section "Key issues in software design"

Key Issues in Software Design	
Sous-sections	Concepts
Concurrency	▪ How to decompose the software into processes, tasks and threads ▪ Efficiency, atomicity, synchronization, scheduling
Control and Handling of events	▪ Organize data and control flow ▪ Events handling
Distribution of components	▪ Distribute the software across hardware ▪ Component communication ▪ Middleware and heterogeneous software
Errors, exception handling Fault Tolerance	▪ To prevent and tolerate faults, exception handling
Interaction and presentation	▪ Structure and organization of the interactions with users and business logic
Data persistence	

Tableau XXVII

Concepts de la section "Software structure and architecture"

Software Structure and Architecture	
Sous-sections	Concepts
Architectural structures and viewpoints	▪ Logical view (functional requirement) ▪ Process view (concurrency) ▪ Physical view (distribution) ▪ Development view (implementation units)
Architectural Styles	▪ General structure, distributed systems, interactive systems, adaptable systems, rule-based systems
Design patterns	▪ Micro architecture ▪ Creational patterns, Structural patterns, Behavioural patterns
Families of programs and frame work	▪ Maximize reuse

Tableau XXVIII

Concepts de la section "Software design quality analysis and evaluation"

Software Design Quality Analysis and Evaluation	
Sous-sections	Concepts
Quality attributes	▪ Discernable at run-time o Performance, security, availability, functionality, usability) ▪ Non-discernable at run-time o Modifiability, portability, reusability, testability, integrability
Quality analysis and evaluation techniques	▪ Software design reviews & inspections ▪ Static analysis ▪ Simulation and prototyping
Measures	▪ Function-oriented design measures ▪ Object-oriented design measures

Tableau XXIX

Concepts de la section "Software Design Notations"

Software Design Notations	
Sous-sections	Concepts
Structural descriptions (static view)	▪ Architecture description languages ▪ Class and object diagrams ▪ Component diagrams ▪ Collaboration responsibilities cards ▪ Deployment diagrams ▪ Entity-relationship diagrams ▪ Interface description languages ▪ Jackson structure diagrams ▪ Structure charts
Behaviour descriptions (dynamic view)	▪ Activity diagrams ▪ Collaboration diagrams ▪ Data flow diagrams ▪ Decision tables and diagrams ▪ Flowcharts and structured flowcharts ▪ Sequence, state transition and statechart diagrams ▪ Formal specification languages ▪ Pseudo-code and program design languages

Tableau XXX

Concepts de la section "Software design strategies and methods"

Software Design Strategies and Methods	
Sous-sections	Concepts
General Strategies	▪ Divide and conquer ▪ Stepwise refinement ▪ Top-down & Bottom-up ▪ Data Abstraction & information hiding ▪ Heuristics use ▪ Patterns use ▪ Iterative and incremental approach
Function-oriented design	▪ Identifying software functions ▪ Data flows diagram
Object-oriented design	▪ Inheritance and polymorphism ▪ Component-based design

Tableau XXX (suite)

Software Design Strategies and Methods	
Sous-sections	Concepts
Data-structure design	▪ Data structure
Component design	▪ Independence ▪ Interfaces ▪ Reuse

3.2.1.3 Construction du logiciel (Software Construction)

Les connaissances du domaine de la construction logiciel concernent les activités de réalisation des composants du logiciel telle la programmation, les tests unitaires et d'intégration et la mise au point. Pour réaliser ces activités, des outils sont utilisés tels des compilateurs, des environnements de programmation, des débogueurs et des utilitaires facilitant les tests des composants développés. Ce domaine est divisé en trois sections présenté au niveau des trois prochains tableaux.

Tableau XXXI

Concepts de la section "Software construction fundamentals"

Software Construction Fundamentals	
Sous-sections	Concepts
Minimizing complexity	▪ Code : simple and readable ▪ Use of standards and quality techniques
Anticipating change	
Constructing for verification	▪ Detect faults ▪ Coding standards ▪ Code review, unit testing, automated testing
Standards in construction	▪ Tools (UML) ▪ External standards o Languages, tools, interfaces o IEEE, ISO, OMG ▪ Internal standards o At corporate level or specific projects

Tableau XXXII

Concepts de la section "Managing Construction"

Managing Construction	
Sous-sections	Concepts
Construction Models	▪ Waterfall models ▪ Iterative models
Construction Planning	▪ Choice of construction method ▪ Reduce complexity, anticipate change, construct for verification
Construction Measurement	▪ Code developed, modified, reused, destroyed ▪ Code complexity and inspection statistics ▪ Fault-fix and fault-find rates

Tableau XXXIII

Concepts de la section "Practical Considerations"

Practical Considerations	
Sous-sections	Concepts
Construction Design	▪ Constraints of the real-world problem
Construction languages	▪ Configuration files ▪ Toolkit languages o Programming languages : linguistic, formal, visual
Coding	▪ Techniques for code writing ▪ Use of classes, enumerated types, variables, constant ▪ Control structures ▪ Handling errors conditions ▪ Code-level security breaches ▪ Resources usage ▪ Source code organization ▪ Code documentation and tuning
Construction Testing	▪ Unit testing ▪ Integration testing ▪ IEEE 829 et 1008

Tableau XXXIII (suite)

Practical Considerations	
Sous-sections	Concepts
Reuse	▪ Integrating reuse processes and activities in life cycle ▪ Selection of reusable units, databases, test procedures ▪ Evaluation of code or test reusability ▪ Reporting reuse
Construction quality	▪ Unit testing and integration testing ▪ Code stepping ▪ Use of assertions ▪ Debugging ▪ Technical reviews ▪ Static analysis
Integration	▪ Routines, classes, components, subsystems ▪ Planning the sequence of integration ▪ Tests

3.2.1.4 Tests du logiciel (Software Testing)

Les tests sont effectués afin d'évaluer la qualité du logiciel et de l'améliorer en identifiant les défauts. Les tests du logiciel se concentrent sur la vérification dynamique du comportement du logiciel à l'aide d'un jeu d'essais. Les activités de tests s'intègrent maintenant plus tôt dans les activités de développement et ne sont plus confinés qu'à la suite de la programmation. Le domaine des tests se subdivise en cinq sections qui sont présentées au niveau des cinq prochains tableaux

Tableau XXXIV

Concepts de la section "Software testing fundamentals"

Software Testing Fundamentals	
Sous-sections	Concepts
Testing-related Terminology	▪ Dynamic verification ▪ Finite set of test cases ▪ Selection criterion ▪ Expected behaviour ▪ Fault & defect: cause ▪ Failure: undesired effect
Keys issues	▪ Test selection criteria ▪ Testing effectiveness ▪ Testing for defect identification ▪ Theoretical and practical limitations of testing ▪ Testability
Relationships of testing with others activities	▪ Quality ▪ Formal Verification ▪ Debugging ▪ Programming ▪ Certification

Tableau XXXV

Concepts de la section "Test levels"

Test Levels	
Sous-sections	Concepts
Target of the test	▪ Unit testing (IEEE 1008) ▪ Integration testing (components) ▪ System testing
Objectives	▪ Conformance, Correctness or Functional testing ▪ Acceptance testing ▪ Installation testing ▪ Alpha, Beta testing ▪ Reliability ▪ Regression and stress testing ▪ Back to back testing ▪ Configuration testing ▪ Usability testing ▪ Test-driven development

Tableau XXXVI

Concepts de la section "Test Techniques"

Test Techniques	
Sous-sections	Concepts
Intuition & Experience	▪ Ad hoc testing ▪ Exploratory testing
Specification-based	▪ Equivalence partitioning ▪ Boundary-value analysis ▪ Decision table ▪ Finite-state machine-based ▪ Formal specifications ▪ Random testing
Code-based	▪ Control-flow based ▪ Data flow based ▪ Reference models o Flowgraph, Call graph
Fault-based	▪ Error guessing ▪ Mutation testing
Usage-based	▪ Operational profile ▪ Software reliability Engineered testing
Nature of Application	▪ Object-oriented ▪ Component based ▪ Web-based ▪ GUI testing ▪ Concurrent testing ▪ Protocol conformance testing ▪ Real-time testing ▪ Safety-critical testing
Selection & combining techniques	▪ Functional & structural ▪ Deterministic vs random

Tableau XXXVII

Concepts de la section "Test related measures"

Test Related Measures	
Sous-sections	Concepts
Evaluation of program	▪ Program size and structure (complexity) ▪ Fault types, classification and statistics ▪ Fault density ▪ Life test ▪ Reliability growth models
Evaluation of the tests	▪ Coverage/thoroughness measures ▪ Fault seeding ▪ Mutation score

Tableau XXXVIII

Concepts de la section "Test Process"

Test Process	
Sous-sections	Concepts
Practical considerations	▪ Attitude / Egoless programming ▪ Test Guide ▪ Test process management o People, tools, policies, measurements, ▪ Test documentation o Test plan o Test procedure specification o Test log o Test incident & problem report ▪ Test cost estimation ▪ Termination ▪ Test reuse and patterns
Tests Activities	▪ Planning ▪ Test cases generation ▪ Test environment development ▪ Test execution ▪ Test results evaluation ▪ Problem reporting ▪ Defect tracking

3.2.1.5 La maintenance du logiciel (Software Maintenance)

La maintenance du logiciel s'intéresse aux activités de modifications du logiciel suite à sa livraison pour effectuer des correctifs et des améliorations ou pour l'adapter à un nouvel environnement. Les activités de maintenance englobent la mise à jour des différents documents du logiciel incluant, entre autres, les guides d'utilisation. La norme ISO/IEC 14764 ajoute les activités de planification des travaux de maintenance. Le domaine de connaissance de la maintenance se subdivise en quatre sections présentées par les quatre prochains tableaux

.

Tableau XXXIX

Concepts de la section " Software Maintenance Fundamentals"

Software Maintenance Fundamentals	
Sous-sections	Concepts
Definitions and Terminology	▪ Modification of a software product after delivery to correct faults, to improve performance or to adapt the product to a modified environment ▪ Modification to code and documentation due to a problem or the need of improvement
Nature of maintenance	▪ Operational life cycle ▪ Modification request are logged and tracked ▪ Testing is conducted ▪ New release of the software ▪ Training and support to users ▪ Maintainer ▪ Problem and modification analysis
Need for maintenance	▪ Correct faults ▪ Improve the design ▪ Implement enhancements ▪ Interface with other systems ▪ Adapt programs ▪ Migrate legacy software ▪ Retire software
Maintenance costs	
Evolution of software	▪ Complexity increases
Categories	▪ *Corrective, adaptive, perfective, preventive*

Tableau XL

Concepts de la section " Key Issues in Software Maintenance"

Key Issues in Software Maintenance	
Sous-sections	Concepts
Technical issues	▪ Limited understanding, Testing, Impact analysis ▪ Maintainability
Management issues	▪ Alignment with organizational objectives ▪ Staffing, Process ▪ Organizational aspects of maintenance ▪ Outsourcing
Maintenance Cost Estimation	▪ Cost estimation, Parametric models, Experience
Software Maintenance Measurement	▪ Benchmarking techniques ▪ Analyzability, Changeability, Stability, Testability

Tableau XLI

Concepts de la section "Maintenance Process"

Maintenance Process	
Sous-sections	Concepts
Maintenance Process	▪ Problem and modification analysis ▪ Modification Implementation ▪ Maintenance review, acceptance ▪ Migration ▪ Software retirement
Activities	▪ Unique activities o Transition o Modification request : acceptance/rejection o Problem report help desk o Software support o Service level agreements and contracts ▪ Software maintenance planning ▪ Concept document for maintenance ▪ Software configuration management ▪ Software Quality

Tableau XLII

Concepts de la section "Techniques for Maintenance"

Techniques for Maintenance	
Sous-sections	Concepts
Program Comprehension	▪ Code browsers ▪ Documentation
Re-engineering	▪ Legacy systems
Reverse Engineering	▪ Identify the components of software ▪ Create high level abstraction ▪ Date reverse engineering

3.2.1.6 Gestion des configurations du logiciel

Ce domaine s'intéresse à l'identification de tous les composants d'un logiciel, incluant les différents documents et plans afin d'être en mesure dans le temps de gérer d'une façon systématique les changements apportés à ceux-ci, tout en conservant leur intégrité. Ce domaine se divise en six sections présentées par les six prochains tableaux.

Tableau XLIII

Concepts de la section "Management of the SCM process"

Management of the SCM process	
Sous-sections	Concepts
Organizational Context	▪ Designated individuals ▪ Closest relationship with development and maintenance
Constraints and Guidance for the SCM process	▪ Corporate policies and procedures ▪ Contract ▪ Best practices ▪ CMMI

Tableau XLIII (suite)

Management of the SCM process	
Sous-sections	Concepts
Planning for SCM	▪ SCM resources and Schedules ▪ Tool selection and implementation o SCM library o Change request and approval procedures o Code and change management tasks o Reporting and SCM measurements o Auditing o Managing and tracking documentation o Performing software builds o Managing and tracking software release ▪ Vendor/Subcontractor control ▪ Interface control
SCM Plan	▪ Introduction (purpose, scope, terms used) ▪ Management (organisation, responsibilities, policies, procedures) ▪ Activities (identification, control) ▪ Resources (tools, resources, human resources) ▪ Maintenance
Surveillance of SCM	▪ SCM measures and measurement ▪ In-process audits

Tableau XLIV

Concepts de la section "Software Configuration Identification "

Software Configuration Identification	
Sous-sections	Concepts
Identifying items to be controlled	▪ Software configuration ▪ Software configuration item and relationships ▪ Software version ▪ Baseline ▪ Acquiring software configuration items
Software Library	▪ Access control, Security,

Tableau XLV

Concepts de la section " Software Configuration Control "

Software Configuration Control	
Sous-sections	Concepts
Requesting, Evaluating, Approving Software Changes	▪ SCM Control Board ▪ Software change request ▪ Deviations and Waivers
Implementing software changes	▪ Software procedures ▪ Tools for tracking changes ▪ Library tools (check-in check-out)
Deviations and Waivers	

Tableau XLVI

Concepts de la section "Software Configuration Status Accounting "

Software Configuration Status Accounting	
Sous-sections	Concepts
SCM Status information	▪ Capture and reporting information ▪ Configurations : identified, collected, maintained
SCM Status reporting	▪ Development, maintenance, project management, SQA teams

Tableau XLVII

Concepts de la section " Software Configuration Auditing "

Software Configuration Auditing	
Sous-sections	Concepts
Functional Audit	▪ Software component consistent with its specifications
Physical Audit	▪ Software design & documentation is consistent to software component
In-process audit	▪ Investigate the current status of specific elements

Tableau XLVIII

Concepts de la section "Software Release Management and Delivery"

Software Release Management and Delivery	
Sous-sections	Concepts
Software building	▪ Combining the correct versions of software configuration items and the appropriate configuration data into an executable program
Software Release management	▪ Identification, packaging and delivery of elements of a product

3.2.1.7 Software Engineering Management

Le thème de la gestion de projet pose un problème. D'une part, il est identifié et décrit au sein d'un chapitre du SWEBOK, combiné avec le management de l'ingénierie du logiciel et d'autre part, il est aussi identifié comme étant une discipline limitrophe du génie logiciel. Au même titre, entre autres, que l'informatique. Nous avons déjà pris position à l'effet que l'informatique n'est du génie logiciel, malgré son influence certaine. De plus, SWEBOK ne consacre pas de chapitre spécifique à l'informatique. Ainsi, les principes candidats contenant des concepts de l'informatique ne seront pas retenus comme étant des principes du génie logiciel. Cependant, qu'en sera-t-il pour les principes contenant des concepts de gestion de projet ? Nous choisissons l'orientation suivante basée sur les choix des auteurs de SWEBOK à l'effet que les concepts généraux de la gestion de projet ne sont pas spécifiques à la gestion de projet logiciel, mais que certains aspects décrits dans le chapitre de SWEBOK sont particuliers au génie logiciel. Nous considérons donc que les concepts généraux répertoriés au sein du *Project Management Body of Knowlewdge (PMI2004)* appartiennent à la discipline de la gestion de projet. Ainsi, PMBOK identifie 44 processus regroupés en neuf domaines tel que présentés au tableau XLIX.

Tableau XLIX

Domaines de connaissances de la gestion de projet (PMBOK-PMI 2004)

1. Project Integration Management	2. Project Scope Management	3. Project Time management
4. Project Cost Management	5. Project Quality Management	6. Project Human Resource management
7. Project communication Management	8. Project Risk Management	9. Project Procurement Management

SWEBOK intitule le chapitre de la gestion de projet comme « *Software engineering Management* » en renforçant les particularités des projets logiciels. Globalement, les cinq processus présentés au sein du chapitre de SWEBOK ne sont pas exclusifs au génie logiciel, cependant, les façons de faire, telles les méthodes, les techniques et les outils utilisés sont spécialisés pour le logiciel. Une nette emphase est constatée au niveau de l'implémentation d'un processus de mesure et de son intégration à la gestion d'un projet.

Les auteurs du chapitre soulignent certaines particularités de la gestion de projet logiciel que nous structurons en fonction des axes suivants : produits, processus et individus.

Le logiciel est un *produit* caractérisé, entre autres, par sa complexité peu perceptible pour les clients. Ainsi, le client perçoit difficilement l'impact sur le produit d'un changement ou d'un ajout, contrairement à un immeuble en construction où l'impact d'un changement peut, en général, se visualiser d'une façon concrète. Le logiciel peut être aussi d'une grande nouveauté, sans équivalent jusqu'à présent, ce qui augmente les facteurs de risques sur le projet. Également, les technologies utilisées pour le développement du logiciel sont caractérisées par un cycle de vie très court qui apportent des contraintes sur le déroulement du projet, telles le manque de soutien technique, de fiabilité, de maturité et de formation des développeurs.

Au niveau du *processus*, celui-ci peut engendrer en cours d'exécution des changements et même la création de nouvelles exigences. À titre d'exemple, suite à la revue d'un prototype avec le client, celui-ci peut demander des changements ou des ajouts qu'il n'avait pas encore envisagés jusque là. Egalement, le processus de développement du logiciel est de nature itérative, plutôt que d'être séquentielle. Le processus est aussi caractérisé par un équilibre délicat entre la créativité et le maintient d'une certaine discipline et de rigueur dans la réalisation des étapes du projet. Le guide SWEBOK (2004) souligne également que la mesure joue un rôle important dans le management de l'ingénierie du logiciel. En effet, ils soulignent que la gestion de projet logiciel sans mesure (qualitative et quantitative) est un manque de rigueur et prive le management d'indicateurs importants sur le déroulement du projet. Ainsi, un projet logiciel doit inclure un processus de prise de mesures en cours de réalisation du projet. Ce processus de mesure est décrit dans la norme ISO15939. Les techniques de mesures utilisées sont spécifiques au génie logiciel.

La gestion des *individus* a aussi des particularités pour le logiciel. Les individus doivent recevoir ou maintenir une formation sur les technologies dont le cycle de vie est très court. Ainsi, les gestionnaires de projet doivent s'assurer que la formation des individus est maintenue à jour. De plus, l'expertise pointue est difficile à obtenir, ainsi, les gestionnaires de projet doivent mettre en place des mesures pour l'acquérir et la retenir au sein de l'organisation. C'est ainsi que le gestionnaire de projet doit trouver un équilibre entre l'autonomie du personnel et conserver un processus discipliné.

Ce chapitre de SWEBOK souligne un autre aspect propre au logiciel, soit la gestion du porte folio des logiciels en développement et en exploitation dans l'organisation. Cette vue globale est non seulement nécessaire pour les arrimages entre les systèmes logiciels, mais aussi pour évaluer les impacts des changements et de détecter les possibilités de réutilisation des composants logiciels.

SWEBOK subdivise le domaine en six sections, les cinq premières traitent du processus de gestion de projet et la dernière concerne la mesure. Le processus de gestion de projet n'est pas particulier au génie logiciel, cependant, nous tenterons d'identifier et de retenir les concepts plus près des particularités du génie logiciel.

Tableau L

Concepts de la section "Initiation and Scope Definition"

Initiation and Scope Definition	
Sous-sections	Concepts
Determination & negotiation of Requirements	▪ Elicitation methods and techniques ▪ Boundaries for the tasks undertaken ▪ Validation and changes procedures
Feasibility analysis	▪ Expertise
Process for the review & Revision of Requirements	▪ Agreement between stakeholders ▪ Revised at predetermined points ▪ Traceability analysis and risk analysis

Tableau LI

Concepts de la section "Software Project Planning"

Software Project Planning	
Sous-sections	Concepts
Process planning	▪ Selection of the appropriate life cycle model ▪ Adaptation and deployment of processes ▪ Selection of tools and methods
Determine deliverables	▪ For each tasks ▪ Reuse of software components evaluated ▪ Use off the shelf software product ▪ Use of third parties
Effort, Schedule and cost estimation	▪ Tasks, inputs and outputs ▪ Expected effort based on historical size-effort data ▪ Iterative activity
Resource allocation	▪ Equipment, facilities and people ▪ Allocation of responsibilities

Tableau LI (suite)

Software Project Planning	
Sous-sections	Concepts
Risk management	▪ Software unique aspects of risk, such adding unwanted features
Quality management	▪ Quality attributes ▪ Product verification and validation
Plan management	▪ Reporting, monitoring and control

Tableau LII

Concepts de la section "Software Project Enactment"

Software Project Enactment	
Sous-sections	Concepts
Implementation of plans	▪ Resources, deliverables
Supplier contract management	▪ Prepare and execute agreements ▪ Monitor performance and accept products
Implementation of measurement process	▪ Relevant data collection
Monitor process	▪ Adherence to the various plans ▪ Outputs for eack tasks analyzed ▪ Effort, schedule and costs are investigated ▪ Measurement data are modeled and analyzed
Control process	▪ Corrective action ▪ Revision of plans ▪ Abandonment of the project ▪ Configuration Management
Reporting	

Tableau LIII

Concepts de la section " Review and Evaluation"

Review and Evaluation	
Sous-sections	Concepts
Determining satisfaction of Requirements	▪ User and customer satisfaction ▪ Variances are identified

Tableau LIII (suite)

Review and Evaluation	
Sous-sections	Concepts
Reviewing and evaluating Performance	▪ Methods, tools, and techniques are evaluated ▪ Process relevance, utility and efficacy

Tableau LIV

Concepts de la section "Closure"

Closure	
Sous-sections	Concepts
Determining Closure	▪ Requirements satisfied ▪ Stakeholders involment
Closure activities	▪ Archival of project materials ▪ Measurement data base updated

Tableau LV

Concepts de la section " Software Engineering Measurement "

Software Engineering Measurement	
Sous-sections	Concepts
Establish and sustain measurement commitment	▪ Accept requirements for measurement ▪ Commit resources for measurement
Plan the measurement process	▪ Characterize the organizational unit ▪ Identify information needs ▪ Select measures ▪ Define data collection, analysis and reporting procedures ▪ Define criteria for evaluating the information products ▪ Review, approve and provide resources for measurement tasks ▪ Acquire and deploy supporting technologies

Tableau LV (suite)

Software Engineering Measurement	
Sous-sections	Concepts
Perform the measurement process	▪ Integrate measurement procedures with relevant processes ▪ Collect data ▪ Analyze data and develop information products ▪ Communicate results
Evaluate measurement	▪ Evaluate information products ▪ Evaluate the measurement process ▪ Identify potential improvements

3.2.1.8 Software Engineering Process

Ce domaine regroupe les connaissances de la gestion de projet de génie logiciel ainsi que les mesures associées aux différents processus. La première section couvre la définition du projet et son envergure, incluant les exigences, les études de faisabilité et le processus de revue des exigences. La deuxième section est la planification du projet de développement du logiciel incluant, entre autres, les livrables, l'estimation de l'effort, des coûts et de l'échéancier. La troisième section couvre la mise en place des différents plans, de la gestion des sous-contractants, mise en place des processus de prise de mesure, de contrôle et de rapport. La quatrième section couvre la revue et l'évaluation. La cinquième section couvre les activités de fin de projet. La dernière section regroupe les connaissances reliées à la prise de la mesure tant au niveau du processus que du produit.

Tableau LVI

Concepts de la section "Process Implementation and Change"

Process Implementation and Change	
Sous-sections	Concepts
Process Infrastructure	▪ Resources (staff, tools, funding) ▪ Responsibilities assigned ▪ Infrastructure o Software engineering process group o Experience factory (models, guides, courses)
Software process management cycle	▪ Establish Process Infrastructure ▪ Planning ▪ Process implementation and change ▪ Process evaluation
Models for process implementation and change	▪ Quality improvement paradigm ▪ IDEAL model ▪ Quantitative or qualitative
Practical considerations	▪ Changes to the process ▪ Changes to the process outcomes (returning investment)

Tableau LVII

Concepts de la section "Process Definition"

Process Definition	
Sous-sections	Concepts
Software Life Cycle Models	▪ Waterfalls, prototyping, evolutionary, incremental iterative, spiral, reusable, automated software synthesis
Software Life cycle processes	▪ Process definition
Notations for process definitions	▪ Data flow diagrams ▪ Statecharts ▪ Actor-dependency modeling ▪ SADT notation ▪ Petri nets ▪ Rule-based
Process adaptation	▪ Local adaptations
Automation	▪ Tolls to support process notations

Tableau LVIII

Concepts de la section "Process Assessment"

Process Assessment	
Sous-sections	Concepts
Process assessment models	▪ Capture the good practices ▪ Maturity model (ex : CMMI)
Process assessment methods	▪ CBA-IPI : process improvement ▪ SCAMPI : CMMI

Tableau LIX

Concepts de la section "Process and Product Measurement"

Process and Product Measurement	
Sous-sections	Concepts
Process measurement	▪ Quantitative information about the process, collected, analysed and interpreted. ▪ Identify the strengths and weaknesses of processes ▪ Evaluate processes after implementation or changes ▪ Productivity of teams
Software Product Measurement	▪ Product size ▪ Product structure ▪ Product quality
Quality of measurement results	▪ Accuracy, reproducibility, repeatability, convertibility, random measurement errors
Software Information models	▪ Models using data collected and knowledge ▪ Model building o Calibration and evaluation of the model ▪ Model implementation o Interpretation and refinement of models o
Process measurement techniques	▪ Process measurement techniques o Analyze software processes and to identify strengths and weaknesses o Instruments based and judgmental ▪ Analytic and benchmarking

3.2.1.9 Outils et méthodes

Ce domaine regroupe les connaissances associées aux outils et aux méthodes utilisées en génie logiciel. La section des outils suit la structure même des chapitres du guide. La deuxième section se concentre sur les méthodes applicables au développement de logiciel.

Tableau LX

Concepts de la section "Software Engineering Tools"

Software Engineering Tools	
Sous-sections	Concepts
Software Requirements Tools	▪ Requirement modeling ▪ Requirements traceability
Software Design Tools	
Software Construction Tools	▪ Program editors ▪ Compilers & code generators ▪ Interpreters ▪ Debuggers
Software Testing Tools	▪ Test generators ▪ Test execution frameworks ▪ Test evaluation ▪ Test management ▪ Performance analysis
Software maintenance Tools	▪ Comprehension ▪ Reengineering
Software CM Tools	▪ Defect enhancement issue and problem tracking ▪ Version management ▪ Release and build
Software Engineering Management Tools	▪ Project planning and tracking ▪ Risk management ▪ Measurement

Tableau LX (suite)

Software Engineering Tools	
Sous-sections	Concepts
Software engineering Process Tools	▪ Process modeling ▪ Process management ▪ Integrated CASE ▪ Process-centered software
Software Quality Tools	▪ Review and audit ▪ Static analysis
Miscellaneous Tools Issues	▪ Tools integration techniques ▪ Meta tools ▪ Tool evaluation

Tableau LXI

Concepts de la section "Software Engineering Methods"

Software Engineering Methods	
Sous-sections	Concepts
Heuristic Methods	▪ Structured Methods ▪ Data Oriented methods ▪ Object oriented methods
Formal Methods	▪ Specification languages and notations ▪ Refinement ▪ Verification/ proving properties
Prototyping Methods	▪ Styles ▪ Prototyping target ▪ Evaluation techniques

3.2.1.10 Qualité du logiciel

Cette section regroupe les connaissances concernant la qualité du logiciel. Et elle recoupe les différentes étapes du développement. La première section élabore les fondements de la qualité du logiciel incluant, entre autres, les modèles, les coûts, les caractéristiques de qualité et l'amélioration de la qualité. La seconde section regroupe les éléments de connaissances reliés à la gestion de la qualité incluant les audits et les

revues. La dernière section présente des considérations pratiques de la qualité du logiciel.

Tableau LXII

Concepts de la section "Software Quality Fundamentals"

Software Quality Fundamentals	
Sous-sections	Concepts
Culture and Ethics	▪ To share a commitment to software quality ▪ Code of ethics to reinforce attitudes related to quality
Value and Cost of quality	▪ Prevention cost, appraisal cost, interval & external failure cost
Models and Quality Characteristics	▪ Software engineering process quality ▪ Software product quality
Quality Improvement	▪ Iterative process of continuous improvement o Management control, coordination, feedback ▪ Prevention and detection of errors ▪ Customer focus

Tableau LXIII

Concepts de la section "Software Quality Management Processes"

Software Quality Management Processes	
Sous-sections	Concepts
Software quality Assurance	▪ Processes, products and resources ▪ Assurance that software products and processes conform to their specified requirements
Verification and Validation	▪ To ensure that quality is built into software ▪ Verification : ensure that the product is built correctly ▪ Validation : ensure that the right product is built ▪ Planning o Assign roles, resources, responsibilities
Reviews and Audits	▪ Management Reviews ▪ Technical reviews ▪ Inspections ▪ Walk-through ▪ Audit

Tableau LXIV

Concepts de la section " Practical Considerations "

Practical Considerations	
Sous-sections	Concepts
Software Quality Requirements	▪ Influence factors o Domain of the system, requirements, standards, budget, staff, etc. ▪ Dependability o In case of system failure ▪ Integrity levels of software o Consequences of failure and probability
Defect Characterization	▪ Error, Fault, Failure, Mistake
Software quality management Techniques	▪ Static Techniques ▪ People-intensive techniques ▪ Analytical techniques ▪ Dynamic techniques ▪ Testing
Software Quality Measurement	▪ Management decision making ▪ Find problematic areas and bottlenecks in process ▪ Testing

3.2.2 Les disciplines frontières du génie logiciel

Le guide SWEBOK identifie huit disciplines frontières avec le génie logiciel. La figure 21 présente ces disciplines.

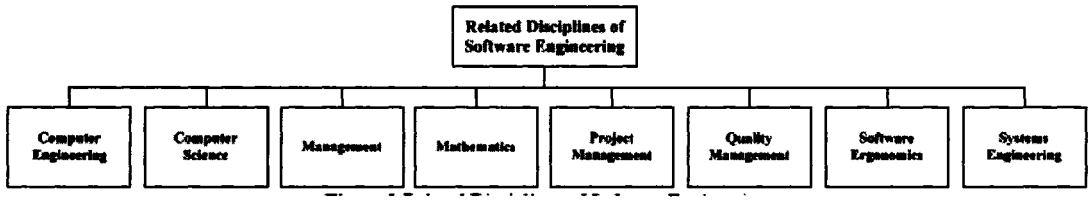

Figure 21 Disciplines frontières du génie logiciel (SWEBOK Abran et al. 2004)

Parmi les disciplines frontières, deux d'entre elles attirent plus notre attention. Le génie informatique et l'informatique sont les disciplines qui ont des frontières très importantes avec le génie logiciel et pas toujours très nettes. Ces deux disciplines ont aussi plusieurs thèmes en commun entre elles. Cependant, le génie informatique est plus marqué par l'aspect matériel des ordinateurs que l'informatique.

Afin d'obtenir certaines balises précises sur les thèmes et les concepts de l'informatique, nous détaillons cette discipline.

3.2.3 L'informatique

L'informatique est une des disciplines qui comporte des liens forts avec le génie logiciel. D'ailleurs, ces liens sont tellement étroits qu'il est parfois difficile de distinguer si un aspect particulier est du génie logiciel ou de l'informatique. Ces zones grises sont une source de discussions entre les membres des ces deux disciplines. De plus, est-ce que le génie logiciel est une sous discipline de l'informatique ou une discipline à part entière? Les discussions à ce sujet sont à la source de plusieurs écrits et conférences.

Pour les besoins de cette recherche, nous avons besoin d'identifier les principaux concepts de l'informatique. Au niveau de la revue de littérature, nous avons observé que plusieurs énoncés de principe comportaient des concepts de l'informatique. Dans le but de raffiner l'analyse de ces principes, l'identification des principaux concepts de l'informatique permettra de mieux distinguer les principes relevant du génie logiciel de ceux de l'informatique.

À l'instar du génie logiciel, les définitions de l'informatique abondent. À notre grande surprise, ni SWEBOK et ni le curriculum de l'informatique (CC2001) nous offrent une définition explicite de l'informatique en tant que discipline. Pour pallier à ce problème,

nous choisissons la définition proposée par « l'Encyclopédie de l'informatique » (Ralston et al.2000) : « *...the systematic study of algorithmic processes that describe and transform information : their theory, analysis, design, efficiency, implementation and application.* » *(p.405)*

Ralston et al.(2000) soulignent que la discipline se divise en deux grandes parties, les *applications informatiques* et les *systèmes informatiques*. Les applications informatiques regroupent les thèmes concernant le traitement de l'information et sa représentation. Les applications se subdivisent en deux groupes. Le premier regroupe les applications de nature numérique où il y a dominance des modèles mathématiques et des données numériques. Le deuxième groupe concerne les applications dites « non-numériques » où les problèmes et l'information sont représentés par des symboles et des règles. Le tableau LXV synthétise les catégories et les thèmes associés de l'informatique.

Tableau LXV

Catégories et thèmes de l'informatique (Adaptation de Ralston et al. (2000))

Applications informatiques	**Systèmes informatiques**
1 - Numériques ▪ Analyse numérique ▪ Méthodes d'optimisation ▪ Simulation ▪ Librairies de fonctions mathématiques ▪ Géométrie	1 - Systèmes logiciels ▪ Programmes ▪ Compilateurs, assembleurs ▪ Traitement des langages intermédiaires ▪ Outils de programmation ▪ Systèmes d'exploitation ▪ Gestion des réseaux télécom.
2 - Non numériques ▪ Intelligence artificielle ▪ Systèmes multimédias ▪ Traitement du langage ▪ Graphiques ▪ SGBD	2 - Systèmes matériels ▪ Design logique ▪ Organisation des ordinateurs ▪ Architecture matérielle ▪ Processeurs, mémoire ▪ Composants physiques

Les systèmes informatiques se subdivisent également en deux catégories : les systèmes logiciels et les systèmes matériels. La catégorie des systèmes logiciels traite, entre autres, des outils de programmation tels les compilateurs, les assembleurs et les systèmes d'exploitation. La catégorie des systèmes matériels traite des thèmes concernant l'organisation et l'architecture matérielle des ordinateurs incluant les configurations des processeurs, de la mémoire, des périphériques.

Ralston et al. (2000) identifient trois sphères principales d'activités : théoriques, expérimentales et de design. Les activités théoriques comprennent la conception de nouvelles théories, de nouveaux modèles et de notations pour mieux comprendre les relations entre les objets d'un domaine. Les activités expérimentales permettent de tester les théories, les modèles et les notations proposés et les raffiner. Les activités de design se concentrent sur la conception de systèmes et d'applications informatiques pour répondre aux besoins d'une organisation ou d'un domaine d'application.

Selon Ralston et al. (2000), le corpus des connaissances de l'informatique se divise en douze sous domaines. Chacun de ces sous domaines sont présentés sommairement en identifiant les concepts principaux. Ces concepts seront utiles lors de l'analyse ultérieure des principes.

Les prochaines sections présentent les concepts des 12 domaines de l'informatique. Le lecteur peut aller, à sa guise, directement à la section 3.2.4.

3.2.3.1 Les algorithmes et les structures de données

Les algorithmes représentent la séquence des étapes d'une solution à un problème. L'algorithme sera, par la suite, traduit dans un langage de programmation qui donnera les instructions à être exécutées par l'ordinateur dans une séquence donnée. L'algorithme est le point de départ de l'automatisation des tâches. L'algorithmique est

l'étude systématique des algorithmes et se divise en trois sphères d'activités : le design pour la conception des algorithmes; l'analyse pour l'évaluation de l'efficacité, de la complexité et des performances et la vérification pour la preuve du bon fonctionnement de l'algorithme.

Une structure de données est une collection organisée de valeurs et un ensemble d'opérations qu'il est possible d'effectuer sur celles-ci. Les structures de données classiques comprennent entre autres, les ensembles, les piles, les files, les listes, les arbres, les graphes, les tableaux et les articles. Le concept d'abstraction est étroitement lié aux structures de données par l'entremise de l'interface des opérations (services) qui offre une abstraction sur l'implantation physique des structures afin de faciliter leur manipulation.

Tableau LXVI

Principaux concepts du domaine des algorithmes et des structures de données (Adaptation de Ralston et al. (2000))

▪ Algorithmes ▪ Structures de donnée o Listes o Arbres o Adressage aléatoire o Graphes ▪ Programmation parallèle ▪ Algorithmes probabilistiques ▪ « Pattern-matching » ▪ Combinatoire ▪ Cryptographie	▪ Diviser pour régner ▪ Programmation dynamique ▪ Interpréteur d'état et de piles ▪ Test ▪ Algorithmes parallèles ▪ Évaluation de la complexité et de la performance ▪ Algorithme aléatoire

3.2.3.2 Les langages de programmation

Le langage de programmation est un outil que le programmeur dispose pour dicter à l'ordinateur la séquence des opérations à être exécutées. Le programmeur peut traduire

un algorithme dans un des nombreux langages de programmation existants. Les langages offrent aussi des possibilités pour organiser le code d'une solution afin d'améliorer la lisibilité et la facilité d'entretien du code. Un langage de programmation est composé de mots et de règles syntaxiques dont le programmeur doit faire l'apprentissage afin de programmer une solution. Les langages évolués offrent une couche d'abstraction sur la configuration matérielle de l'ordinateur. Le compilateur, l'interpréteur ou la machine virtuelle se chargent de la traduction des programmes en un code binaire exécutable par l'ordinateur. Les langages se classent selon leur paradigme. Ainsi, les paradigmes les plus connus sont : procédural, orienté-objet et fonctionnel. En 1999, on pouvait répertorier plus de 1000 langages de programmation commerciaux, académiques ou expérimentaux (Ralston et al. (2000)).

Tableau LXVII

Principaux concepts du domaine des langages de programmation (Adaptation de Ralston et al. (2000))

▪ Modèles d'ordinateur o Machine de Turing ▪ Génération & traduction ▪ Langages formels ▪ Sémantique, syntaxe, grammaire ▪ Expression régulière ▪ Types et objets ▪ Simulation ▪ Graphiques ▪ Compilateurs, assembleurs ▪ Analyseur syntaxique ▪ Débogueur, Parseur	▪ Type statique & dynamique ▪ Fonctionnelle ▪ Objet ▪ Procédurale ▪ Flux de données ▪ Graphiques ▪ Décomposition fonctionnelle ▪ Types abstraits de données ▪ Abstraction ▪ Traitement de donnée

3.2.3.3 Architecture des ordinateurs

L'architecture des ordinateurs est le domaine de l'informatique qui traite de l'organisation physique des composants de l'ordinateur. Ce domaine a des liens étroits

avec la microélectronique ainsi qu'avec les mathématiques. L'architecture comporte deux volets. L'architecture système se préoccupe d'offrir aux programmeurs une vue fonctionnelle et abstraite sur la configuration matérielle. L'implantation de l'architecture traite des aspects de coûts et de performances des composants matériels. L'architecture décompose l'ordinateur en trois groupes de composants : les processeurs, le stockage de l'information (mémoire, disques) et les entrées-sorties incluant les communications entre les composants de l'ordinateur. Ce domaine intègre aussi la gestion des erreurs et la tolérance aux pannes.

Tableau LXVIII

Principaux concepts du domaine de l'architecture des ordinateurs (Adaptation de Ralston et al. (2000))

▪ Logique digitale ▪ Algèbre booléenne ▪ Mathématiques discrètes et statistiques ▪ «Finite state theory » ▪ Théorie des nombres ▪ Gestion des erreurs ▪ Tolérance aux pannes	▪ Structures de contrôle ▪ Optimisation ▪ Organisation des ordinateurs ▪ Simulateurs ▪ Entrée-sortie ▪ Parallélisme ▪ « Finite state machine » ▪ Architecture multiprocesseurs

3.2.3.4 Les systèmes d'exploitation et les réseaux

Les systèmes d'exploitation sont des logiciels spécialisés dont leur but est de gérer les ressources matérielles de l'ordinateur. Ces logiciels offrent une abstraction sur l'architecture complexe des différents composants physiques d'un ordinateur en offrant une gamme de services de haut niveau qui permettent d'interagir avec l'ordinateur. Les thèmes de ce domaine sont, entre autres, les aspects de sécurité, la fiabilité, les files d'attente, l'ordonnancement, la concurrence, les inter blocages, la tolérance aux pannes

et la gestion des erreurs. Au niveau des réseaux, on y retrouve les thèmes des protocoles, la gestion distribuée, la bande passante et la nomination des ressources.

Tableau LXIX

Principaux concepts du domaine des systèmes d'exploitation et des réseaux (Adaptation de Ralston et al. (2000))

▪ Gestion de la concurrence ▪ Ordonnancement ▪ Gestion de mémoire ▪ Réseaux ▪ Analyse et modèles de performance ▪ Systèmes en temps réel ▪ Protocoles réseaux	▪ l'abstraction ▪ «Information hiding » ▪ Encapsulation ▪ «Binding » ▪ Gestion de processus et de tâches ▪ Gestion de mémoire ▪ Gestion de fichiers

3.2.3.5 Le génie logiciel

Ralston et al. (2000) identifient le génie logiciel comme un domaine de l'informatique. Le génie logiciel se concentre sur la réalisation de grands systèmes logiciels qui répondent aux besoins exprimés tout en étant fiables et sécures. Comme nous avons déjà choisi SWEBOK (2004) comme référence pour l'identification des concepts du génie logiciel, nous ne ferons pas d'identification de concepts dans cette section.

3.2.3.6 Les systèmes de gestion de bases de données

Ce domaine regroupe les thèmes reliés aux bases de données et à la gestion sécuritaire de grands ensembles de données. Les SGBD sont des logiciels qui font la gestion des informations en permettant de les rechercher, de les consulter et de les mettre à jour en en préservant leur intégrité. Les SGBD intègrent aussi des gestionnaires de transactions qui permettent de mettre à jour les données d'une façon intègre et sécuritaire. Ce domaine comporte également les thèmes de sécurité des accès aux données, l'encryptage

de l'information, l'optimisation des requêtes, la sauvegarde et le recouvrement des données.

Tableau LXX

Principaux concepts du domaine des bases de données (Adaptation de Ralston et al. (2000))

▪ Algèbre relationnelle ▪ Concurrence ▪ Sérialisation des transactions ▪ Prévention des deadlocks ▪ Synchronisation ▪ Inférence statistique ▪ Inférence basée sur des règles ▪ Techniques de tris et de recherches ▪ Indexation ▪ Analyse de performance ▪ Cryptographie ▪ Authentification	▪ Modélisation de données ▪ Fichiers ▪ Accès aux données ▪ Optimisation des requêtes ▪ Contrôle de la concurrence et du recouvrement ▪ Intégrité ▪ Sécurité ▪ Machine virtuelle ▪ Intégration multimédia

3.2.3.7 L'intelligence artificielle et la robotique

Le domaine de l'intelligence artificielle traite de la modélisation des aspects cognitifs des humains dans le but de concevoir des machines et des logiciels capables de les reproduire ou de les augmenter. Ce domaine comporte trois volets. La *psychologie informatique* fait l'étude du comportement et de l'intelligence humaine afin de concevoir des programmes qui seraient en mesure d'imiter le raisonnement humain. La *philosophie informatique* fait l'étude de la constitution d'une base de connaissances pouvant être traitées par l'ordinateur. Le dernier volet, *l'intelligence de la machine*, fait l'étude de méthodes de programmation permettant aux ordinateurs et aux robots d'exécuter des tâches dont seuls les humains étaient en mesure de faire. Ce domaine comporte aussi l'étude des thèmes suivants : la reconnaissance des sons, de la parole et du langage; des images, des patterns; de l'apprentissage et du raisonnement.

Tableau LXXI

Principaux concepts du domaine de l'intelligence artificielle et de la robotique (Adaptation de Ralston et al. (2000))

▪ Logique	▪ Robotique
▪ Règles	▪ Représentation des connaissances
▪ Connaissance	▪ Résolution de problèmes
▪ Méthodes formelles	▪ Heuristiques
▪ Méthodes de recherches	▪ Apprentissage
▪ Théories d'apprentissage	▪ Compréhension du langage
▪ Reconnaissance de la parole	▪ Réseaux de neurones

3.2.3.8 Les graphiques par ordinateur

Le domaine des graphiques par ordinateur s'intéresse aux modèles pour représenter les graphiques en deux ou trois dimensions. La réalité virtuelle et la simulation (aussi sous forme de jeux) sont des applications concrètes des théories et des modèles développés dans ce domaine. Deux volets importants composent ce domaine : la représentation (modèle interne) des images et les méthodes pour afficher efficacement en considérant les aspects de performances. De plus, les animations et l'interaction des utilisateurs avec les images sont aussi des thèmes couverts par le domaine. Les fondements théoriques s'appuient, entre autres, sur la géométrie, les mathématiques et les algorithmes.

Tableau LXXII

Principaux concepts du domaine des graphiques par ordinateur (Adaptation de Ralston et al. (2000))

▪ Géométrie	▪ « Smoothing »
▪ Caos	▪ Fractales
▪ Graphique	▪ Animation
▪ Échantillonnage	▪ Réalité virtuelle
▪ Simulation	▪ Gestion des couleurs

3.2.3.9 Les interactions personne-machine

Ce domaine traite de la coordination des interactions et des informations échangées entre l'humain et l'ordinateur. Ce domaine a des liens importants avec les graphiques et les interfaces personne-machine ainsi qu'avec les sciences cognitives. Les aspects d'ergonomie, de sécurité, d'apprentissage ainsi que les périphériques tels les souris, les claviers et autres capteurs de sens sont aussi des thèmes étudiés par le domaine. La conception des interfaces utilisateurs est un volet important de ce domaine puisqu'elles comptent pour beaucoup dans la facilité d'utilisation d'un logiciel. La réalité virtuelle est une application concrète dans laquelle les interactions personne machine jouent un rôle important.

Tableau LXXIII

Principaux concepts du domaine des interactions personne machine (Adaptation de Ralston et al. (2000))

▪ Science cognitive ▪ Analyse de risque ▪ Ergonomie	▪ Structure de données ▪ Traitement des images ▪ Panneau de contrôle

3.2.3.10 Le calcul numérique

Ce domaine fait l'étude de phénomènes scientifiques inaccessibles sans une capacité de calcul élevée offerte par des ordinateurs de grande puissance de traitement numérique. Les scientifiques et les ingénieurs utilisent la puissance de calcul pour résoudre des équations complexes et valider des modèles complexes. Sans l'aide des ordinateurs, les calculs complexes demandant de grande quantité d'itérations ne pourraient se faire par les personnes durant leur vie. Ce domaine était nommé « traitement numérique » jusqu'au début des années 1980. Les fondements de ce domaine s'appuient fortement sur les mathématiques ainsi que sur l'algorithmique.

Tableau LXXIV

Principaux concepts du domaine du calcul numérique (Adaptation de Ralston et al. (2000))

▪ Mathématiques ▪ Analyse d'erreurs ▪ Géométrie ▪ Statistiques ▪ Quantum ▪ Intégration symbolique	▪ Approximations ▪ Erreurs ▪ Stabilité ▪ Élément fini ▪ Convergence ▪ Parallélisme

3.2.3.11 Les systèmes d'information

Ce domaine traite des aspects de l'automatisation des tâches et des processus d'affaires au sein d'une entreprise ou organisme. Un système d'information se définit comme une collection de personnes, de procédures et d'équipements afin de recueillir, d'enregistrer, de retrouver, de traiter et d'afficher l'information de gestion. Comme l'automatisation des tâches et des processus d'affaires est considérée névralgique, ce domaine a des liens étroits avec d'autres disciplines telles les sciences de la gestion et le management, l'ingénierie des systèmes et les interfaces personne-machine. Les fondements de ce domaine sont, entre autres, l'organisation du travail, la gestion et la modélisation des processus.

Tableau LXXV

Principaux concepts du domaine des systèmes d'information (Adaptation de Ralston et al. (2000))

▪ Langages ▪ Systèmes d'exploitation ▪ Réseaux ▪ Bases de données ▪ Intelligence artificielle	▪ Interface personne – machine ▪ Linguistique ▪ Sciences de la gestion ▪ Sciences cognitives

3.2.3.12 La bioinformatique

La bioinformatique est un domaine en pleine émergence. Ce domaine représente une collaboration étroite de l'informatique au développement des disciplines de la biologie et de la médecine. L'informatique permet, entre autres, de représenter et de valider certains modèles de ces disciplines d'une façon visuelle et rapidement. L'étude de la chaîne de l'ADN et de ces composants est un exemple où l'informatique est d'un grand support, de même pour l'étude des structures chimiques des enzymes. De plus, l'étude de composants de mémoire organique fait l'objet de collaboration entre l'informatique et la biologie, tout comme la conception d'implants permettant de restaurer l'audition et la vue chez les humains.

Tableau LXXVI

Principaux concepts du domaine de la bioinformatique (Adaptation de Ralston et al. (2000))

▪ Algorithmes ▪ Traitement des images ▪ Graphiques ▪ Architecture des ordinateurs	▪ Structure de données ▪ Calcul numérique ▪ Combinatoire ▪ Bases de données ▪ Robotique

3.2.4 Les activités du génie logiciel (ISO/IEC 12207)

Pour faire suite à l'identification des domaines de connaissances du génie logiciel et de leurs principaux concepts, il est utile pour notre recherche d'identifier les principaux processus et activités du génie logiciel. À cet effet, nous faisons référence à la norme ISO/IEC 12207 (1995). Cette norme internationale établit un cadre commun, incluant une terminologie normalisée, sur les processus, les activités et les tâches associés au cycle de vie du logiciel. Un volet intéressant de cette norme est l'attention donnée à l'acquisition de composant logiciel ou de services. Ainsi, ceci s'ajoute aux activités

traditionnelles de développement, d'opération et de maintenance du logiciel. La norme couvre les situations suivantes :

- L'acquisition d'un système comprenant un composant logiciel
- L'acquisition ou la fourniture d'un logiciel ou d'un service logiciel
- Le développement, l'opération et la maintenance du logiciel

La norme regroupe un ensemble de processus comprenant pour chacun des activités et des tâches spécifiques à réaliser tout au long du cycle de vie. Pour les besoins de cette recherche, nous ne décrirons pas en détail tout le contenu de la norme, mais notre attention portera sur l'identification des principaux processus et sur les activités principales qui y sont associées.

La norme regroupe les processus du cycle de vie du logiciel sous trois catégories : primaire, de soutien et organisationnel. Le tableau LXXVII dresse la liste des processus selon les catégories.

Tableau LXXVII

Catégories de processus (adaptation de ISO/IEC 12207)

Processus du cycle de vie ISO/IEC 12207		
Primaires	**De soutien**	**Organisationnels**
▪ Acquisition ▪ Approvisionnement ▪ Développement ▪ Exploitration ▪ Maintenance	▪ Documentation ▪ Gestion des configurations ▪ Assurance qualité ▪ Vérification ▪ Validation ▪ Revue ▪ Audit ▪ Résolution des problèmes	▪ Gestion de projet ▪ Infrastructure ▪ Amélioration ▪ Formation

3.2.4.1 Processus primaires

La catégorie « primaires » regroupe cinq processus et 35 activités principales concernant l'acquisition, la fourniture, le développement, l'exploitation et la maintenance du logiciel. Le tableau LXXVIII suivant identifie les principales activités associées à chacun des processus primaires.

Tableau LXXVIII

Principales activités associées aux processus primaires (adaptation de ISO/IEC12207)

1. Processus d'acquistion	
▪ Démarrage ▪ Appel d'offre ▪ Préparation du contrat	▪ Suivi des fournisseurs ▪ Acceptation
2. Processus d'approvisionnement	
▪ Démarrage ▪ Préparation d'appel d'offre ▪ Contrat ▪ Planification	▪ Exécution et suivi ▪ Revue et évaluation ▪ Livraison
3. Processus de développement	
▪ Démarrage ▪ Analyse des exigences système ▪ Architecture du système ▪ Analyse des exigences logicielles ▪ Conception architecturale du logiciel ▪ Conception détaillée ▪ Programmation et tests	▪ Intégration logicielle ▪ Test de qualification de logiciel ▪ Intégration système ▪ Test de qualification du système ▪ Installation du logiciel ▪ Acceptation du logiciel

Tableau LXXVIII (suite)

4. Processus d'exploitation	
▪ Démarrage ▪ Test opérationnel	▪ Opération du système ▪ Soutien aux utilisateurs
5. Processus de maintenance	
▪ Démarrage ▪ Analyse des problèmes et des modifications ▪ Mise en place des modifications	▪ Revue et acceptation ▪ Migration ▪ Retrait du logiciel

3.2.4.2 Processus de soutien

Les processus de soutien viennent compléter les processus « primaires » en contribuant à la qualité et au succès du projet. Il y a huit processus principaux dans cette catégorie englobant 25 activités principales. Le tableau LXXIX suivant identifie les principales activités associées à chacun des processus de soutien.

Tableau LXXIX

Principales activités associées au processus de soutien (adaptation de ISO/IEC12207)

1. Processus de documentation	
▪ Démarrage ▪ Conception et développement	▪ Rédaction ▪ Mise à jour
2. Processus de gestion des configurations	
▪ Démarrage ▪ Identification des éléments ▪ Contrôle de la configuration	▪ Suivi de la configuration ▪ Évaluation de la configuration ▪ Gestion des versions et des livraisons

Tableau LXXIX (suite)

3. Processus d'assurance qualité	
▪ Démarrage ▪ Conformité du produit	▪ Conformité du processus ▪ Conformité du système de qualité
4. Processus de vérification	
▪ Démarrage	▪ Contrats ▪ Des processus ▪ Exigences, du design ▪ Code, intégration et documentation
5. Processus de validation	
▪ Démarrage	▪ Cas de tests ▪ Conformité des tests avec les exigences ▪ Effectuer les tests ▪ Valider la conformité du logiciel ▪ Tester le logiciel dans son environnement
6. Processus de revue conjointe	
▪ Démarrage	▪ Du code par rapport au design ▪ De la documentation ▪ Des tests avec les exigences ▪ De l'exécution des activités ▪ Des coûts et des échéanciers
7. Processus d'audit	
▪ Démarrage ▪ Revue de suivi de projet	▪ Revues techniques
8. Processus de résolution des problèmes	
▪ Démarrage	▪ Description du problème ▪ Analyse du problème ▪ Solution au problème ▪ Mise en place

3.2.4.3 Processus organisationnels

La catégorie des processus organisationnels regroupe les processus utilisés par l'entreprise pour structurer l'organisation des projets et les ressources humaines impliquées. Cette catégorie comprend quatre processus principaux et 14 activités principales. Le tableau LXXX suivant identifie les principales activités associées à chacun des processus de soutien.

Tableau LXXX

Principales activités associées aux processus organisationnels (adaptation de ISO/IEC12207)

1. Processus de management	
▪ Démarrage ▪ Planification ▪ Déroulement et suivi	▪ Revue et évaluation ▪ Fin du projet
2. Processus d'infrastructure	
▪ Démarrage ▪ Mise en place de l'infrastructure	▪ Mise à jour de l'infrastructure
3. Processus d'amélioration	
▪ Mise en place ▪ Évaluation	▪ Amélioration
4. Processus de formation	
▪ Démarrage ▪ Conception du matériel de formation	▪ Mise en place du plan de formation

3.3 Définition des termes concept et principe

L'une des premières constatations soulignées à la lecture des travaux antérieurs sur les principes du génie logiciel, est l'absence fréquente de définition des termes utilisés. Ainsi, en plus d'être à la source d'une certaine confusion dans l'utilisation de ces termes, il en découle également l'absence de critères qui permettraient de mieux identifier ce qu'est un principe, par exemple. Dans cette section, les termes « concept », « principe » et « loi » sont définis à l'aide d'une synthèse basée sur l'analyse de plusieurs définitions données à ces termes.

3.3.1 Qu'est ce qu'un concept?

L'encyclopédie Universalis (2002) souligne que l'être humain a essentiellement deux modes de représentation des connaissances. Le premier mode est le concret qui porte sur la connaissance en lien direct avec un objet du monde réel. L'autre mode est l'abstrait qui est représenté par la connaissance par « concepts ». Les concepts se distinguent par leur caractère abstrait et universel, en plus d'avoir la propriété d'être relativement stables. L'aspect universel est repris par Wikpedia en ajoutant qu'un même concept peut s'exprimer en plusieurs langues par des termes différents, mais conserver la même signification. Le concept permet d'unifier les représentations des objets et des idées.

Les concepts peuvent être de deux types : théorique ou observationnel. (Universalis) Les concepts théoriques concernent les propriétés non-observables tandis que les concepts observationnels concernent les objets observables de la réalité physique.

On a recensé 13 définitions utiles du terme concept qui sont répertoriées à l'annexe 1. Comme plusieurs de celles-ci partagent des points communs, on retient, en premier lieu, la définition donnée par Le Robert (2002) : « *Représentation mentale et générale d'une idée, d'un objet, d'une notion générale* ».

Bunge (2003) ajoute qu'un concept est : « *Simple idea, unit of meaning, building block of a proposition. Every concept can be symbolised by a term* »

Le concept, représenté par un terme, est le composant de base de toute proposition. Une proposition est définie comme étant « *un énoncé qui exprime une relation entre deux ou plusieurs termes* » (Le Robert 2002). Le concept serait l'unité de connaissance de base avec lequel les propositions telles, entre autres, les principes et les lois sont formulées.

La définition que nous proposons pour le terme concept est la suivante :

Une représentation mentale et générale d'une idée, d'un objet ou d'une notion. Le concept, pouvant se symboliser par un terme, est l'unité de base de composition d'une proposition.

3.3.2 Qu'est ce qu'un principe?

Le terme *principe* trouve ses origines latines dans le terme *principium* signifiant le commencement ou l'origine. À cet effet, le premier sens donné au terme principe fait référence aux origines, causes premières ou élément constituant. (Le Robert (2002), Webster (1989)). Les principes sont à la base d'une discipline, d'un phénomène ou du fonctionnement d'un objet.

Un deuxième sens donné au terme principe est « *une proposition première, posée et non-déduite* » (Le Robert (2002)). Une proposition est un énoncé qui exprime une « *relation entre deux ou plusieurs concepts* ». Ainsi, un principe serait d'abord une proposition composée de concepts. De ce fait, un concept à lui seul ne pourrait être considéré comme un principe. Une proposition non-déduite signifie qu'un principe ne peut être déduit d'un autre principe. S'il est déduit d'un autre principe, il ne pourrait alors être qualifié de fondamental. Concernant le qualificatif « fondamental », nous soulignons que

si un principe est une proposition première non déduite, il ne serait plus approprié, dès lors, d'utiliser l'expression « principes fondamentaux » qui serait alors considérée comme un pléonasme.

En plus d'être une proposition, un principe représente aussi une règle, une loi ou une vérité générale et non démontrée (Le Robert (2002), Bunge (2003), Webster (1989)). Une règle est définie comme *« une prescription pour faire quelque chose »* (Le Robert (2002)). Le sens donné au terme règle comprend trois interprétations. La première concerne toute la dimension des règles sociales ou des règlements d'une communauté. Cette interprétation est moins pertinente pour notre recherche. La deuxième concerne les règles scientifiques basées sur les lois de la physique ou de la nature. Le logiciel ne se basant pas vraiment sur ce type de lois, cette interprétation ne sera probablement pas retenue. La troisième concerne les *règles empiriques* adoptées suite à des essais fructueux en pratique. Ces règles empiriques sont à la base de l'action. Cette interprétation est pertinente puisque les principes du génie logiciel seraient à la source de l'action et à la base des pratiques. De plus, les principes peuvent être à la base des processus (activités) et des normes (Moore1998). Cette interprétation sera retenue.

Nous avons noté dans certains travaux antérieurs que le terme *loi* est utilisé à l'occasion pour signifier un principe. Également, il est fréquemment identifié comme un synonyme du terme principe (Le Robert (2002), Webster (1989)). Selon ces mêmes dictionnaires, le terme loi représente un ensemble de règles obligatoires établies par l'autorité afin d'orienter les comportements des individus dans une société. Également, le terme loi peut avoir le sens de représenter un rapport constant entre des phénomènes (lois scientifiques). Ces deux sens donnés au terme loi sont moins pertinents pour l'étude du génie logiciel. Le Robert(2002) présente un troisième sens à l'effet qu'une loi représente *« une règle impérative exprimant un idéal ou une norme. »* Ce sens est directement relié avec la notion de règle présentée précédemment.

Bunge (2003) et Nadeau (1999) ajoutent qu'une loi peut être de nature empirique. Bunge souligne que *« Most if not all of the law-statements in emergent sciences are empirical generalizations. Some of these are precursors of law-statement in the strict sense, which are typical of advanced sciences."* (p.161). Nadeau présente les lois empiriques sous deux écoles de pensée. D'une part, les *inductivistes* considèrent qu'une loi empirique est une « *généralisation vraie susceptible d'être confirmée par leurs exemples positifs* » (p.382). D'autre part, les *déductivistes* considèrent qu'une loi empirique *« prend forme d'un énoncé universel dénotant une régularité empirique et dont il est possible de déduire, en conjonction avec des conditions initiales, des conséquences observables »* (p.382). Une loi empirique n'est pas une vérité absolue au même titre qu'un principe. Ainsi, une loi empirique pourrait être contredite. De plus, une loi empirique est moins générale qu'un principe, ainsi, sa portée serait plus limitée qu'un principe. Cependant, une loi peut se généraliser et être élevée au rang de principe.

Le concept de *loi empirique* sera retenu et isolé du terme principe. À cette étape, on fait l'hypothèse que certains principes recensés pourraient être plutôt des lois empiriques. Cette hypothèse est basée sur le fait que le génie logiciel est une discipline en émergence (dans le sens de Bunge) et que beaucoup de généralisations ont été faites à partir d'observation des pratiques de l'industrie. De ces observations et généralisations, des normes, des méthodes, des lois et des principes ont été suggérés. Ainsi, nous faisons l'hypothèse que le génie logiciel a des principes, mais aussi des lois empiriques. Cependant, cette recherche se concentre toujours sur les l'identification des principes, mais laisse une ouverture à catégoriser certains principes comme étant plutôt des lois empiriques. Ainsi au niveau de la définition du terme principe, on ne retient pas comme tel le terme loi au sens large, mais on conserve l'hypothèse que le génie logiciel peut comporter des lois empiriques.

Pour revenir à la composition de la définition du terme principe, une *vérité non démontrée* trouve son sens dans un synonyme : *axiome*, signifiant une proposition vraie

indémontrable, mais évidente. Le Robert (2002) ajoute qu'un principe est une proposition non-démontrée, mais *vérifiable dans ses conséquences*. Ainsi, il serait possible d'observer dans les résultats, si un principe ou un groupe de principes a été suivi ou non. Litré (2004) affirme que *l'omission d'un principe mène à l'erreur*. Ces erreurs peuvent être observées dans les résultats d'un projet logiciel. De plus, un principe peut se vérifier par l'expérience, donc dans la pratique.

Ainsi, suite à l'exposé des divers éléments des définitions recensées, nous proposons la définition suivante pour le terme principe:

Proposition fondamentale de la discipline formulée sous forme prescriptive (règle), à la source des actions, pouvant être vérifiée dans ses conséquences et par l'expérience.

Pour la suite, nous conservons donc trois entités pour la suite de la recherche représentées par la figure 22.

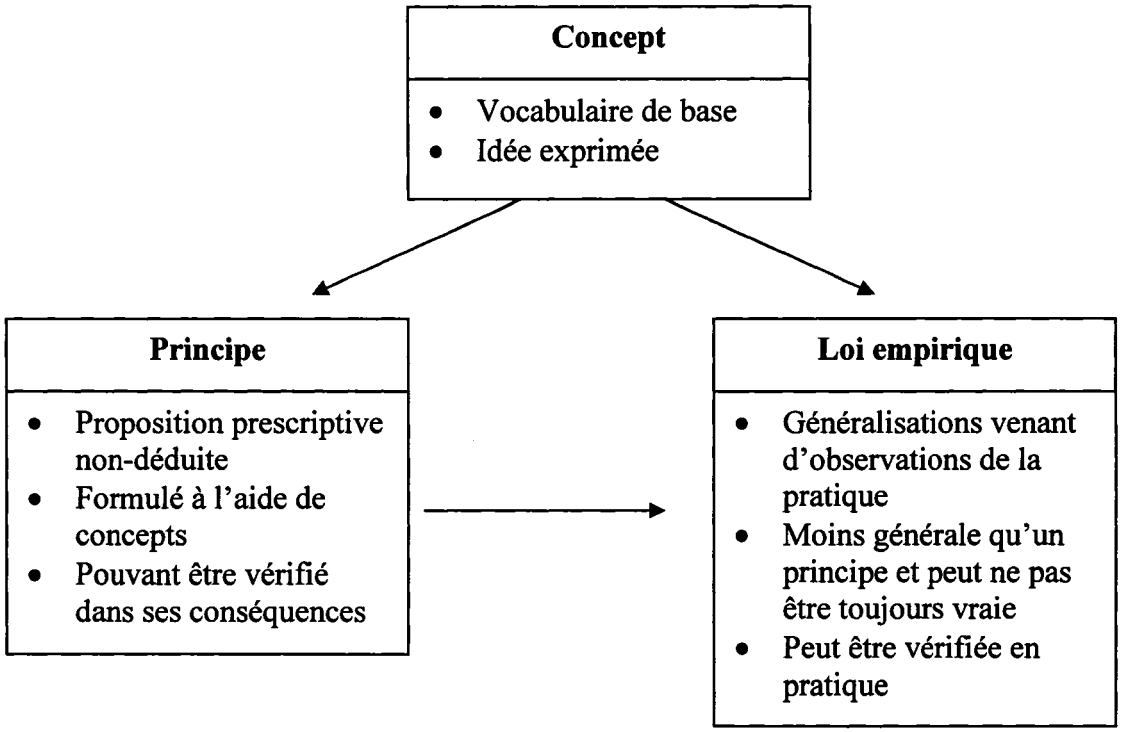

Figure 22 Relations entre les termes

3.4 Critères d'identification des principes

Les critères d'identification sont un composant majeur du cadre conceptuel de cette recherche. Les critères retenus serviront à évaluer chacun des énoncés qui ont été recensés au niveau des travaux antérieurs et de décider s'ils sont conservés ou écartés.

Parmi les travaux antérieurs, seulement un auteur et un groupe d'auteurs ont explicitement spécifié des critères d'identification d'un principe. Il est à noter que c'est seulement en 1983 qu'un auteur a suggéré les premiers critères. Ainsi, Boehm (1983) propose les deux critères suivants :

- *"They should be independent."*

- *"The entire space should be representable by combination of the basic principles."*

Le premier critère est en lien direct avec la définition proposée à l'effet qu'un principe est une proposition non-déduite. Un principe ne peut être déduit d'un autre principe, ainsi, il devrait donc être indépendant. Ce critère ne peut être appliqué lors de la première évaluation du bassin de principes, car un énoncé pris individuellement ne peut satisfaire à ce critère. Cependant, ce critère sera utilisé par la suite comme critère pour évaluer l'ensemble des principes conservés suite à la première évaluation. Cependant, ce critère aura une portée limitée, les interprétations possibles d'une proposition de principe peuvent rendre difficile d'affirmer qu'un principe est indépendant d'une façon *absolue*. Cependant, il serait envisageable de dresser une forme de hiérarchie de principes

Le deuxième critère proposé souligne que l'ensemble des principes devrait couvrir l'ensemble des activités du génie logiciel. Ce critère n'est pas un critère d'identification individuelle ou d'ensemble, mais plutôt une finalité du processus. Ainsi, ce critère ne sera pas retenu pour cette recherche. Il est possible que le groupe de principes retenus suite à l'analyse faite ne couvre pas l'ensemble des activités du génie logiciel.

Le groupe d'auteurs Bourque et al. (2002) propose huit critères d'identification des principes du génie logiciel. Nous présentons chacun des critères proposés en les commentant.

1. "*Fundamental principles are less specific than methodologies and techniques.*"
   La formulation proposée se rapproche plus d'une caractéristique (ou d'une propriété d'un principe que d'un critère d'identification. Cependant, une reformulation mineure pourrait le transformer en un critère utile. Ainsi, la formulation suivante est suggérée :
   Un principe ne doit pas être associé ou découler directement d'une technologie, ou d'une méthode.

De plus, à cette formulation, on ajoute qu'un principe ne doit pas découler d'une technique ou être une activité du génie logiciel tel que défini par la norme IEEE 12207. Cependant, une technique, une technologie, une méthode ou une activité peuvent découler d'un ou plusieurs principes.

2. "*Fundamental principles are more enduring than methodologies and techniques. Principles should be phrased in a way that will stand the test of time rather than in the context of current technology.*"
   Ce critère est lié au premier et représente également une caractéristique d'un principe ou une propriété intrinsèque d'un principe. Compte tenu de la reformulation suggérée du critère no.1, le critère no.2 est intégré à celui-ci et n'est pas retenu comme critère individuel.

3. "*Fundamental principles are typically discovered or abstracted from practice and should have some correspondence with best practices.*"
   L'objectif premier de cette recherche n'est pas de découvrir de nouveaux principes, mais d'évaluer rigoureusement plus de 300 principes déjà identifiés dans les travaux antérieurs. Ainsi, ce critère indique les sources possibles des principes et comment ceux-ci sont en relation avec les pratiques jugées les meilleures de l'industrie. Ce critère ne sera pas retenu.

4. "*Software engineering fundamental principles should not contradict more general fundamental principles.*"
   Ce critère tel que formulé n'est pas applicable. Les *« more general fundamental principles"* ne sont pas identifiés par les auteurs. Ainsi, il n'est pas possible de vérifier concrètement s'il y a contradiction entre les principes éventuellement retenus et les principes généraux non-identifiés. Le critère ne peut être retenu sous sa formulation originale. Cependant, en le reformulant, il peut être retenu comme un

critère d'évaluation de l'ensemble des principes retenus. La formulation suggérée serait : *un principe ne doit pas contredire un autre principe connu.*

5. "*A fundamental principle should not conceal a tradeoff.*"
   La formulation d'un principe ne doit pas suggérer ou imposer un dosage entre deux éléments spécifiés dans l'énoncé. Ces éléments peuvent être entre autres des attributs de qualité du logiciel. Ainsi, les compromis ne peuvent être dictés au niveau des principes, mais ils doivent plutôt être considérés au niveau du processus de développement. Ainsi, ce critère n'est pas retenu.

6. "*...but, there may be tradeoffs in the application of fundamental principles.*"
   Ce critère souligne que l'application d'un principe peut amener à faire des compromis. Ce critère ne permet pas d'identifier ou d'écarter des principes comme tel. Ainsi, il ne peut nous aider à évaluer un principe dans le cadre de la recherche. Le critère ne sera donc pas retenu.

7. "*A fundamental principle should be precise enough to be capable of support or contradiction.*"
   La formulation de ce critère est quelque peu ambiguë. Cependant, notre interprétation est à l'effet que la formulation du principe doit être assez précise pour être en mesure de tester le principe en pratique et de le vérifier dans ses conséquences pratiques. À titre d'exemple, un principe tel « abstraction » ne serait pas une formulation assez précise permettant de vérifier avec précision les conséquences du principe en pratique.

8. "*A fundamental principle should relate to one or more underlying concepts.*"
   Ce critère rejoint des éléments importants de la définition donnée au terme principe. Le principe doit être une proposition contenant un ou des concepts du génie logiciel ou du génie en général. Ainsi, c'est dans ce but que nous avons fait antérieurement

un recensement des concepts du génie logiciel tiré du guide SWEBOK. De plus, nous avons aussi identifié les principaux concepts du génie et de l'informatique. Le critère sera retenu.

Parmi les critères présentés par Boehm (1983) et Bourque et al. (2002) , nous observons deux catégories : les critères *individuels* d'identification et les critères *d'ensemble*. Le tableau LXXXI présente les critères retenus.

Tableau LXXXI

Liste des critères retenus

Critères d'identification individuels

1. Un principe est une proposition formulée de façon prescriptive
   - Indique quoi faire, mais sans spécifier comment le faire
2. Un principe ne doit pas être associé directement ou découler d'une technologie, d'une méthode ou d'une technique ou être une activité du génie logiciel. (adapté de Bourque et al.(2002))
3. Le principe ne doit pas énoncer un compromis (ou un dosage) entre deux actions ou concepts (Bourque et al.(2002) et de Moore (1998))
4. Un principe du génie logiciel devrait inclure des concepts reliés à la discipline ou au génie. (Bourque et al.(2002))
5. La formulation d'un principe doit permettre de le tester en pratique ou de le vérifier dans ses conséquences. (Bourque et al.(2002))
   - Vérifiable dans ses conséquences et par l'expérience

Critères d'ensemble

1. Les principes devraient être indépendants (non déduit) (Boehm (1983))
2. Un principe ne doit pas contredire un autre principe connu. (Bourque et al. (2002))

CHAPITRE 4

PHASE 2 – ANALYSE SELON LES CRITÈRES INDIVIDUELS

4.1 Introduction

Au chapitre 3, nous avons identifié les composants du cadre conceptuel d'analyse des principes. Ainsi, les concepts et les activités du génie logiciel ont été identifiés; les concepts généraux du génie et ceux de l'informatique ont été identifiés; les définitions ont été données aux termes principe, loi et concept; et les critères d'identification des principes ont été identifiés. Ces éléments constituent les principaux composants du cadre conceptuel d'analyse de cette recherche et sont schématisés à la figure 23.

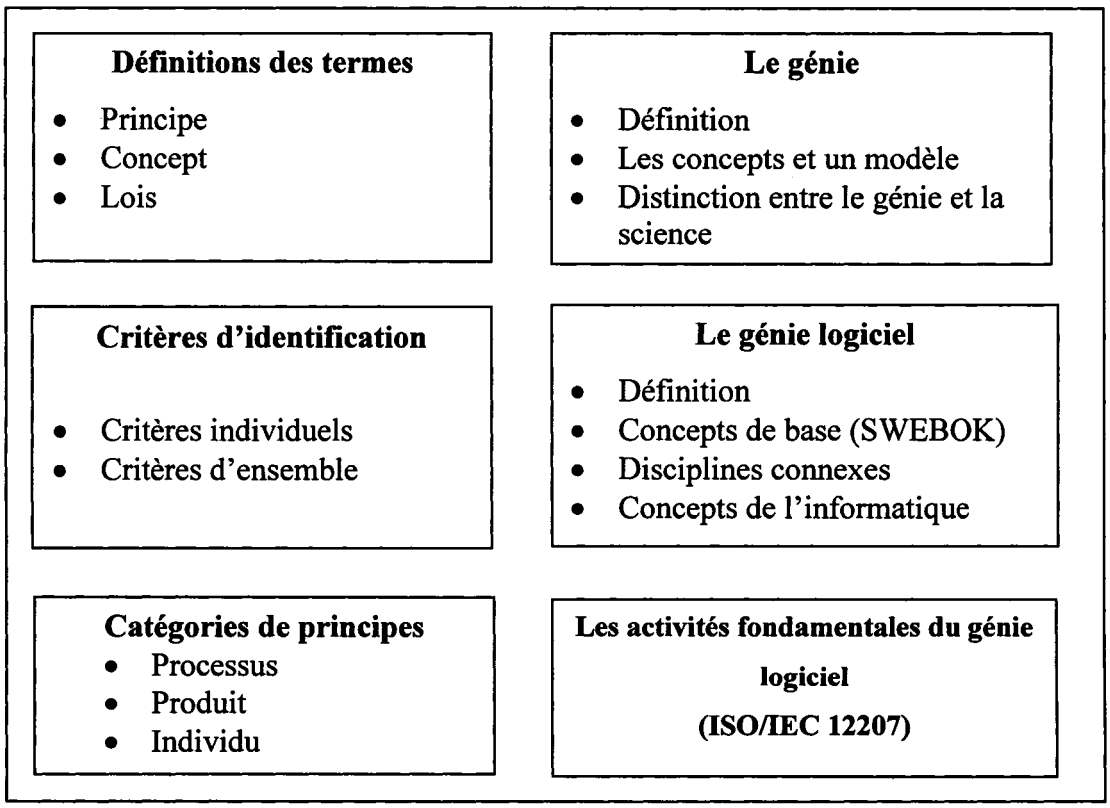

Figure 23 Éléments du cadre conceptuel d'analyse

4.2 Objectifs de la phase 2

L'objectif premier de cette phase est de confronter chacun des principes recensés aux critères d'identification et d'éliminer ceux qui ne satisfont pas aux cinq critères individuels. La réalisation de cette phase va produire une fiche individuelle pour chacun des principes candidats qu'ils soient retenus ou non; un tableau synthèse de l'analyse des principes pour chacun des auteurs et une liste consolidée des principes satisfaisant aux cinq critères d'identification pour l'ensemble des auteurs.

4.3 Méthode de recherche utilisée

La méthode de recherche utilisée consiste à évaluer chacun des principes en fonction des cinq critères d'identification individuels. Ces critères sont supportés par les activités du génie logiciel (ISO/IEEE 12207), les concepts du génie logiciel (SWEBOK), les concepts généraux du génie et les concepts de l'informatique. La figure 24 présente la méthode de recherche suivie pour la phase 2.

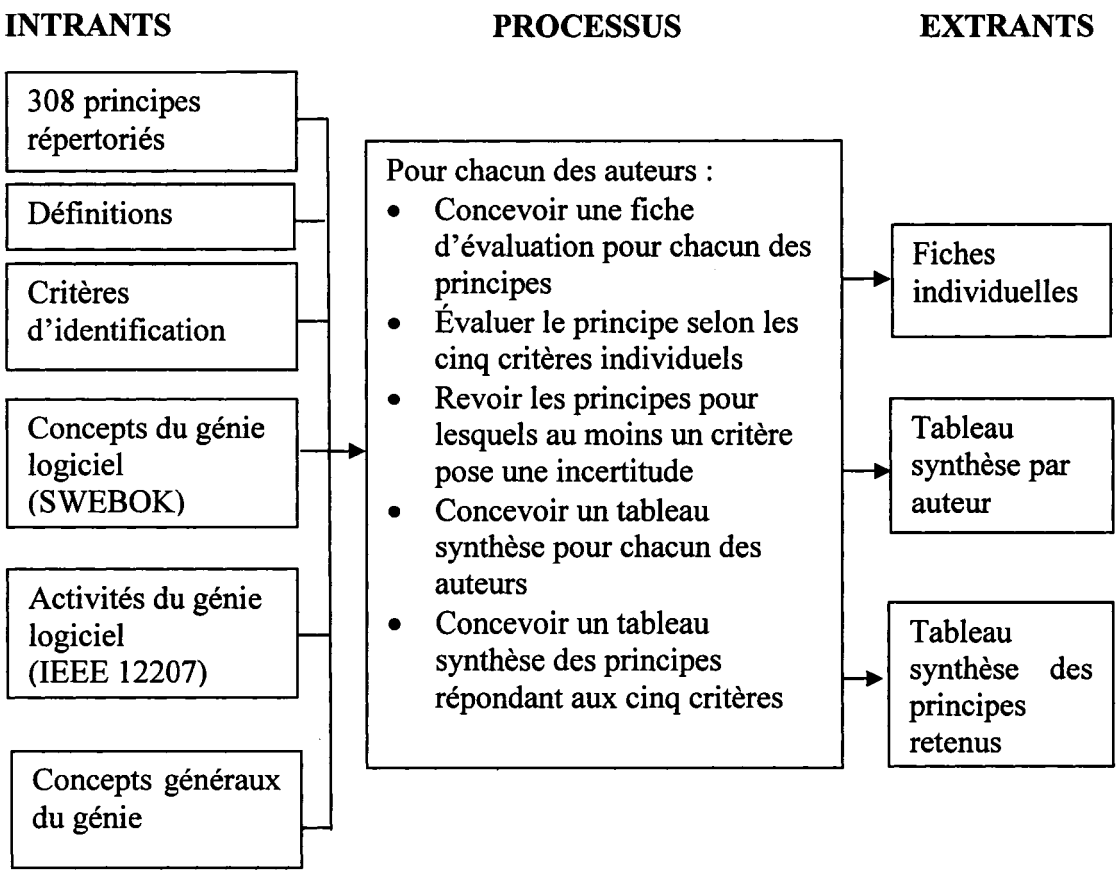

Figure 24 Méthode de recherche de la phase 2

Pour l'étude des principes d'un auteur, deux séries de tableaux seront produits. En premier lieu, une fiche individuelle est conçue pour chacun des principes. Cette fiche est produite dans le but de conserver une trace de l'analyse individuelle des principes. Cette fiche comprend :

- Le nom de l'auteur
- La formulation du principe tel que donné par l'auteur
- Les explications ou commentaires donnés par l'auteur sur le principe, si disponibles

un critère est en rouge, cela signifie que le principe ne satisfait pas le critère et en noire lorsque le critère est satisfait. En cours d'analyse, une codification temporaire « orange » a été utilisée à l'occasion pour identifier les critères et principes qui nécessitaient une recherche plus approfondie soit dans SWEBOK, soit dans les activités de l'ISO/IEEE 12207 ou soit pour évaluer comment un principe pourrait être testé dans ses conséquences. L'utilisation de ce code n'a été que temporaire; à la suite d'un second tour d'analyse, les codes orange ont tous été transformés dans l'un des codes présentés au tableau LXXXII.

Tableau LXXXII

Codification utilisée dans les tableaux synthèses

Symbole	**Signification**
X	Le critère est satisfait
	Le critère n'est pas satisfait (rouge)
	Identification préliminaire d'une loi candidate du génie logiciel (bleue)

La figure 25 montre un exemple du format type d'une fiche individuelle.

Fiche individuelle
Davis 1995

186. Software's entropy increases

Explication de l'auteur

- Any software system that undergoes continuous change will grow in complexity and will become more disorganized.
- This is Manny Lehman's Law of Increasing Entropy

	Commentaires
Critère 1	Formulation non prescriptive
Critère 2	Ne découle pas d'une technologie, technique, d'une méthode ou activité
Critère 3	Pas de compromis
Critère 4	Configuration management : concept du génie logiciel
Critère 5	Formulation testable et vérifiable

Résultat:

- Non retenu

Figure 25 Format type d'une fiche individuelle

Les fiches individuelles sont regroupées en annexe, compte tenu de leur nombre. Ces fiches offrent une traçabilité du processus d'analyse de chacun des principes et celles-ci pourraient être reprises par d'autres chercheurs pour explorer des aspects non couverts par cette recherche.

En second lieu, un tableau synthèse est produit afin de visualiser le résultat de l'analyse pour l'ensemble des principes proposés par un auteur. Ce tableau présente une grille comprenant chacun des principes proposés par l'auteur et la satisfaction ou non des critères individuels. La figure 26 montre le format type du tableau synthèse par auteur.

Synthèse de l'analyse
Davis (1995)
201 principles of Software Engineering
Chapitre 9 – Evolution Principles

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
186	Software's entropy increases		X	X	X	X	
187	If it ain't broke, don't fix it.	X	X	X		X	NON
188	Fix problems, not symptoms	X	X	X	X	X	OUI

Figure 26 Format type du tableau synthèse pour un auteur

Un numéro séquentiel est assigné à chacun des principes, si l'auteur ne l'a pas fait. Les critères sont identifiés par un numéro, tel que défini au chapitre des définitions et critères d'identification. Une colonne « retenu » indique si le principe a été retenu ou non. La codification utilisée est la même que celle présenté au tableau LXXXII.

Un des composants majeurs du processus d'analyse est les critères d'identification des principes élaborés au chapitre précédent. À titre de rappel, le tableau LXXXIII montre les cinq critères individuels utilisés pour la phase 2.

Tableau LXXXIII

Critères individuels d'identification des principes

1. Un principe est une proposition formulée de façon prescriptive
2. Un principe ne doit pas être associé directement ou découler d'une technologie, d'une méthode ou d'une technique ou être en soi une activité du génie logiciel.
3. Le principe ne doit pas dicter un compromis (ou un dosage) entre deux actions ou concepts
4. Un principe du génie logiciel devrait inclure des concepts reliés à la discipline ou au génie.
5. La formulation d'un principe doit permettre de le tester en pratique ou de le vérifier dans ses conséquences.

Le premier critère, à l'effet qu'un principe est une *proposition prescriptive*, a été appliqué d'une façon ferme. Ainsi, la formulation des principes est évaluée comme présentée par chacun des auteurs, aucune reformulation d'un énoncé n'a été effectuée. Le principe doit spécifier l'action à faire, sans pour autant spécifier comment le faire. Compte tenu qu'un principe doit être une *proposition prescriptive*, tous les principes proposés ne comportant qu'un seul mot, tel « encapsulation » ou « abstraction » ne satisfont pas le premier critère.

Le deuxième critère est principalement soutenu par la norme ISO/IEC 12207 qui regroupe l'ensemble des activités du génie logiciel. Ainsi, si un principe est en fait une activité déjà identifiée par la norme, le principe n'est pas retenu. De même, si le principe fait référence à une technologie, une technique ou une méthode, celui-ci n'est pas retenu. Il est à noter qu'un principe peut engendrer des activités, des techniques ou des méthodes. Cependant, le principe en lui même ne devrait pas être une activité, une technique ou une méthode, puisque le principe est au-dessus de celles-ci en termes de portée.

Le troisième critère stipule que la formulation du principe ne peut comporter de dosage ou de compromis entre deux concepts ou actions. Ce critère n'a été utilisé que très occasionnellement. Ainsi, peu de formulations de principe contreviennent à ce critère.

Le quatrième critère stipule que le principe doit contenir, en premier lieu, des concepts du génie logiciel ou, le cas échéant, des concepts généraux du génie. Ainsi, une formulation d'un principe qui ne contient pas de concept explicite du génie logiciel ou du génie, n'est pas retenue. Principalement, le guide SWEBOK (2004) fournira la base des concepts du génie logiciel et le chapitre sur les concepts à la base du génie fournira les concepts généraux du génie. Cependant, le génie logiciel a des frontières avec des disciplines connexes telles, entre autres, l'informatique. Les concepts provenant des

disciplines connexes ne sont pas considérés comme étant du génie logiciel, ni même du génie, à l'exception du management de l'ingénierie, dont SWEBOK réserve un chapitre.

Le cinquième critère dicte que le principe doit être testable et vérifiable dans ses conséquences. Ce critère est plus difficile à appliquer compte tenu que plusieurs propositions de principe sous-entendent les conséquences. Ainsi, il faut déterminer si la variable dépendante (i.e. ce qui pourrait s'améliorer ou se détériorer dans l'application ou non du principe) peut être identifiée dans la formulation du principe ou dans l'explication fournie par l'auteur. À titre d'exemple, le principe *« Product assurance is not a luxury »*, la variable dépendante ne peut être identifiée clairement, le principe ne peut donc pas satisfaire au critère. Également, lors de l'évaluation de ce critère, on a examiné s'il était possible de concevoir une expérimentation, par exemple, deux projets dont un met en œuvre le principe et l'autre non, et que l'on puisse observer les effets de l'application ou non du principe.

Il est à noter que les deux critères d'ensemble seront appliqués à la phase 3 puisqu'ils ne peuvent pas s'appliquer simultanément avec les critères individuels.

4.4 Résultats de la phase 2

Cette section présente les résultats de la phase 2 du processus d'analyse de l'ensemble des principes recensés pour chacun des auteurs et des groupes d'auteurs. Pour chacun des auteurs, seul le tableau synthèse sera présenté dans cette section. Les fiches individuelles se retrouvent en annexe.

4.4.1 Winston W. Royce (1970)

Royce a été dans les premiers auteurs à exprimer une notion de principe à suivre en génie logiciel. L'auteur présente cinq propositions. Parmi celles-ci, une seule proposition

sera retenue. Le tableau LXXXIV présente la synthèse de l'analyse des propositions de Royce.

Tableau LXXXIV

Synthèse de l'analyse des principes de Royce (1970)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Program design comes first	X		X	X	X	NON
2	Documentation must be current and complete	X		X	X	X	NON
3	Do the job twice if possible	X		X		X	NON
4	Testing must be planned, controlled and monitored	X		X	X	X	NON
5	Involve the customer	X	X	X	X	X	OUI

Constat

Les propositions 1,2,3 et 4 ne satisfont pas le critère no. 2. Ces propositions sont directement associées soit à des méthodes, à une des phases du cycle de vie du logiciel ou à des activités prévues du génie logiciel (ISO/IEC 12207). L'énoncé 3 ne satisfait pas au critère no. 4, ne comportant aucun concept explicite du génie logiciel. Seule la proposition 5 satisfait aux cinq critères individuels. En premier lieu, l'énoncé est prescriptif; il ne découle pas d'une activité ou d'une méthode; ne comporte pas de compromis; le client n'est pas un concept spécifique au génie logiciel, mais il est un concept important du processus de génie; et enfin le principe peut être testé en pratique.

Au moment où Royce a écrit ces propositions, les méthodes et les activités du génie logiciel n'avaient pas encore atteint un certain niveau de maturité. Les activités essentielles au développement du logiciel n'étaient pas encore bien identifiées, comme c'est maintenant le cas avec la norme ISO/IEC 12207. Ainsi, les propositions de Royce

sont majoritairement orientées vers des ajustements à faire au cycle de vie en cascades et vers l'identification des activités à faire, plutôt que des principes fondamentaux.

4.4.2 Ross, Goodenough et Irvine (1975)

Ross et al. (1975) identifient sept principes, explicitement nommés comme étant « les » principes du génie logiciel. Les auteurs ne précisent pas la provenance des principes identifiés, mais affirment que ceux-ci ont déjà été identifiés par les « *observateurs* » de la discipline, sans en préciser plus détails. Aucune définition du terme principe n'est donnée, ni de critères d'identification de ceux-ci. Les principes proposés par les auteurs sont plutôt des concepts et non des propositions prescriptives. Le tableau LXXXV présente les résultats de l'analyse.

Tableau LXXXV

Synthèse de l'évaluation des principes de Ross et al. (1975)

		Critères individuels d'identification					
	Principes	#1	#2	#3	#4	#5	Retenu
1	Modularity			X			NON
2	Abstraction		X	X			NON
3	Information hiding			X			NON
4	Localization			X	X		NON
5	Uniformity		X	X	X		NON
6	Completeness		X	X			NON
7	Confirmability		X	X			NON

Constat

Aucun des principes proposés par les auteurs n'est formulé de façon prescriptive. Ainsi, le critère no. 1 n'est pas satisfait pour aucune des propositions. Les principes ne sont que des concepts ou des techniques et non des *propositions prescriptives*. Les principes proposés par les auteurs soutiennent les quatre attributs de qualité qu'un logiciel doit comporter soit : « *modifiability, efficiency, reliability, understandability* ». (Ross et

al.1975) Les principes proposés ne sont pas des propositions guidant l'action dans le développement du logiciel. Également, les sept propositions sont difficilement testables puisque leurs conséquences sont difficilement identifiables. Même en tant que concepts, ceux-ci ne sont tous associés au domaine des concepts du génie logiciel. À titre d'exemple, l'*abstraction* est un concept général et l' « *information hiding* » est une technique de l'informatique. Aucune des sept propositions des auteurs n'est donc retenue.

4.4.3 H.D. Mills (1980)

Mills (1980) identifie sommairement trois principes au niveau du design. L'auteur ne propose pas de méthode pour aborder le thème des principes, ni de définition et de critères d'identification des principes. Le tableau LXXXVI présente les résultats de l'analyse effectuée.

Tableau LXXXVI

Synthèse de l'analyse des principes de Mills (1980)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Sequential process control, characterized by structured programming and program correctness			X	X	X	NON
2	System and data structuring, characterized by modular decomposition			X		X	NON
3	Real-time and multiple/distributed processing control, characterized by concurrent processing and process synchronization			X			NON

Constat

Aucune des propositions n'est retenue. D'une part, aucune ne représente une proposition formulée de façon prescriptive. Ainsi, le critère no. 1 n'est pas satisfait pour aucune des propositions. Également, les propositions sont plutôt associées à des technologies et des techniques; ainsi aucune proposition ne satisfait le critère no. 2. Les concepts contenus dans les propositions sont plutôt associés au domaine de l'informatique. Aucune des trois propositions de Mills n'est donc retenue.

4.4.4 Many Lehman (1980)

Lehman (1980) porte attention sur l'évolution des grands systèmes informatiques. Lehman a analysé différentes données historiques provenant de projets de maintenance sur une période de sept ans. De ses analyses, l'auteur a observé certaines constantes qu'il associe à des lois ou à des principes de base de l'évolution des systèmes. Le tableau LXXXVII présente la synthèse de l'analyse.

Tableau LXXXVII

Synthèse de l'analyse des lois de Lehman

		Critères d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	The law of continuing change		X	X	X		
2	The law of increasing complexity		X	X	X		
3	The fundamental law of large-program evolution		X	X	X		NON
4	The law of organizational stability		X	X			NON
5	The law of conservation of familiarity		X	X			NON

Constat

Aucune proposition n'est retenue parmi les cinq proposées par Lehman (1980). En premier lieu, aucune proposition n'est formulée de façon prescriptive. Le critère no. 1

n'est donc pas satisfait. Les cinq propositions satisfont les critères no. 2 et no. 3. Au niveau du critère no. 4, les propositions 4 et 5 ne comportent pas explicitement de concepts du génie logiciel. Concernant, le critère no. 5, aucune proposition ne satisfait ce critère, compte tenu que la variable dépendante ne peut être clairement identifiée pour chacune des propositions. Nous notons que les deux premières propositions de Lehman (1983) peuvent être considérées comme des lois empiriques potentielles. Dans le premier cas, le logiciel est sujet aux changements afin de l'adapter à l'évolution de son environnement et ce jusqu'à son retrait. Dans le deuxième cas, Lehman souligne que la complexité du logiciel augmente dans le temps. Les changements faits au logiciel amènent une dégradation de sa structure originale dans le temps, ce qui augmenterait sa complexité à effectuer la maintenance.

4.4.5 Barry W. Boehm (1983)

Boehm (1983) identifie sept principes qui seraient à la base du génie logiciel. L'auteur ne définit pas explicitement le terme principe, mais ses propositions sont toutes formulées de façon prescriptive, comme des règles d'action. Le tableau LXXXVIII présente la synthèse de l'analyse faite sur les principes de Boehm.

Tableau LXXXVIII

Synthèse de l'analyse des principes de Boehm (1983)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Manage using phased life cycle plan	X		X	X	X	NON
2	Perform continuous validation	X		X	X	X	NON
3	Maintain disciplined product control	X		X	X	X	NON
4	Use modern programming practices	X		X	X		NON

Tableau LXXXVIII (suite)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
5	Maintain clear accountability for results	X	X	X	X	X	OUI
6	Use better and fewer people	X	X	X	X	X	OUI
7	Maintain a commitment to improve the process	X		X	X	X	NON

Constat

Toutes les propositions sont formulées de façon prescriptive et satisfont donc le critère no. 1. Cependant, cinq propositions sur les sept sont directement associées à des activités prévues à la norme ISO/IEC 12207. Ainsi, le critère no. 2 n'est pas satisfait pour cinq propositions. Toutes les propositions satisfont les critères no. 3 et no. 4. Au niveau du critère no. 5, une proposition n'est pas retenue faute d'identifier la variable dépendante (conséquences). Le tableau LXXXIX présente les détails des principes retenus.

Tableau LXXXIX

Principes retenus de Boehm (1983)

Principes	**Commentaires sur les critères**
Maintain clear accountability for results	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Concepts reliés à la gestion de projet logiciel 5. Formulation vérifiable en pratique
Use better and fewer people	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. La sélection, la composition et l'évaluation du personnel au sein d'un projet logiciel fait partie des concepts reliés à la gestion de projets logiciels. 5. Formulation vérifiable en pratique

Nous constatons, que la majorité des propositions de principes de Boehm sont plutôt des activités à faire dans le processus de développement. Ces activités sont effectivement importantes et elles doivent être faites. Ces activités peuvent être qualifiées de niveau tactique et elles pouvaient ne pas avoir été clairement identifiées par les normes à l'époque où l'auteur les a proposées comme des principes. Aujourd'hui, la norme ISO/IEC 12207 identifie clairement ces groupes d'activités.

4.4.6 Booch et Bryan (1984)

Ce groupe d'auteurs présente au sein d'un chapitre de leur ouvrage, sept principes du génie logiciel. Cependant, les auteurs ne définissent pas, le terme principe et aucun critère d'identification n'est proposé. De plus, les auteurs sont muets sur la méthode suivie pour soutenir le choix de ces principes. Cependant, nous constatons une ressemblance notable avec les propositions de principes de Ross et al. (1975) présentés antérieurement. Le tableau XC présente les résultats de l'analyse.

Tableau XC

Synthèse de l'analyse des principes de Booch et Bryan (1984)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Abstraction		X	X			NON
2	Information hiding			X			NON
3	Modularity			X			NON
4	Localization			X	X		NON
5	Uniformity		X	X	X		NON
6	Completeness		X	X			NON
7	Confirmability		X	X			NON

Constat

Comme les sept principes proposés par les auteurs sont les mêmes que ceux présentés par Ross et al. (1975), les constatations seront les mêmes que celles présentées antérieurement. Ainsi, aucun des principes proposés par les auteurs n'est formulé de façon prescriptive. Aucune des sept propositions des auteurs n'est donc retenue.

4.4.7 Buschmann et al. (1996)

Buschmann et al. (1998) ont proposé onze principes associés à l'architecture du logiciel. Les auteurs ne définissent pas le terme principe, n'identifient aucun critère et ne soulignent pas la méthode utilisée pour arriver à ces résultats. Les auteurs affirment que les principes proposés sont aussi des *techniques* (« *enabling techniques* ») utilisées dans la conception et la construction du logiciel. Ainsi, les termes principe et technique sont considérés comme étant des synonymes par les auteurs. Le tableau XCI présente les résultats de l'analyse.

Tableau XCI

Synthèse de l'analyse des principes de Buschmann et al. (1996)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Abstraction		X	X			NON
2	Encapsulation			X			NON
3	Information hiding			X			NON
4	Modularization			X			NON
5	Separation of concerns		X	X	X		NON
6	Coupling and cohesion			X	X		NON
7	Sufficiently, completeness and primitiveness		X	X	X		NON
8	Separation of policy and implementation			X	X		NON
9	Separation of interface and implementation			X	X		NON
10	Single point of reference		X	X			NON
11	Divide and conquer (decomposition)	X		X	X		NON

Constat

Aucun des principes proposés par les auteurs ne satisfait aux cinq critères d'identification du cadre d'analyse. Dix des onze principes proposés ne sont pas des propositions prescriptives. Sept principes sont en fait associés à des techniques de design. Cinq énoncés ne contiennent aucun concept explicite du génie logiciel ou du génie. Aucun des principes proposés ne peut être testés et vérifiés dans ses conséquences, la variable dépendante ne pouvant être identifiée. Ainsi aucune proposition de principe n'est retenue.

4.4.8 Alan Davis (1995)

Davis (1995) propose un recueil regroupant 201 principes du génie logiciel. Davis est un des premiers auteurs de notre revue à définir le terme principe, cependant, il reste muet sur les critères d'identification utilisés. L'auteur souligne que ses principes ne sont pas indépendants et que des contradictions ou des chevauchements peuvent survenir entre les 201 principes proposés. Davis a structuré sa présentation selon les huit thèmes suivants :

1. Principes généraux
2. Principes des exigences logicielles
3. Principes du design
4. Principes de codage (programmation)
5. Principes des tests
6. Principes de gestion
7. Principes d'assurance qualité
8. Principes d'évolution

Pour l'analyse des principes, nous retiendrons la structure de l'auteur. Au chapitre de la revue de littérature, nous avons souligné que Davis avait produit un article (Davis 1994) présentant les 15 principes les plus importants tirés des 201 principes de son ouvrage. Comme ces 15 principes sont présents dans le recueil, les 15 principes présentés dans la

publication de l'auteur en 1994 ne seront pas traités, compte tenu de la redondance évidente.

4.4.8.1 Principes généraux de Davis

Davis a proposé 37 principes de catégorie générale. L'analyse faite n'en retient que huit tel que présenté au tableau XCII.

Tableau XCII

Synthèse de l'analyse des principes généraux de Davis (1995)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Quality is #1		X	X	X		NON
2	Quality is in the eyes of the beholder		X	X	X		NON
3	Productivity and Quality are inseparable		X	X	X	X	NON
4	High-Quality Software is possible		X	X	X		NON
5	Don't try to retrofit quality	X	X	X	X	X	OUI
6	Poor reliability is worse than poor efficiency		X	X	X		NON
7	Give product to customers early	X	X	X	X	X	OUI
8	Communicate with customers/users	X	X	X	X	X	OUI
9	Align incentives for developer and customer	X	X	X	X	X	OUI
10	Plan to throw one away	X	X	X	X		NON
11	Build the right kind of prototype	X		X	X	X	NON
12	Build the right features into prototypes quickly	X		X	X	X	NON
13	Build throwaway prototypes quickly	X		X	X	X	NON

Tableau XCII (suite)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
14	Grow systems incrementally	X	X	X	X	X	OUI
15	The more seen, the more needed		X	X		X	
16	Change during development is inevitable		X	X	X		
17	If possible, buy instead of build	X		X			NON
18	Build software so that it needs a short user manual	X	X	X	X	X	OUI
19	Every complex problem has a solution.		X	X	X		NON
20	Record your assumptions	X		X		X	NON
21	Different languages for different phases			X	X		NON
22	Technique before tools		X	X	X	X	NON
23	Use tools, but be realistic	X	X		X		NON
24	Give software tools to good engineers	X	X	X	X	X	OUI
25	CASE tools are expensive			X	X		NON
26	Know when is as important as know how		X		X		NON
27	Stop when you achieve your goal	X	X	X			NON
28	Know formal methods	X		X	X	X	NON
29	Align reputation with organization	X	X	X			NON
30	Follow the lemmings with care	X	X	X			NON
31	Don't ignore technology	X	X	X		X	NON
32	Use documentation standards	X	X	X	X	X	OUI
33	Every document needs a glossary			X	X	X	NON
34	Every software document needs an index			X	X	X	NON
35	Use the same name for the same concept	X		X	X	X	NON
36	Research then transfer does not work		X	X		X	NON
37	Take responsibility		X	X			NON

Constat

Seize des 37 principes généraux proposés ne sont pas formulés de façon prescriptive et ne satisfont pas, ainsi, le critère no. 1. Également, onze principes découlent d'activités déjà prévues à la norme ISO/IEC 12207 et ne satisfont pas le critère no. 2. Deux principes comportent dans leur formulation une forme de dosage ou de compromis. Neuf principes ne contiennent pas explicitement de concepts du génie logiciel. Également, 16 propositions ne peuvent être testées ou vérifiées dans leurs conséquences, du fait que l'identification de la variable dépendante pose un problème. Ainsi, les huit principes présentés au tableau XCIII sont retenus :

Tableau XCIII

Principes généraux de Davis retenus

Principes	**Commentaires sur les critères**
Don't try to retrofit quality	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. La qualité est un concept du génie logiciel qui ne peut pas être au produit fini 5. Formulation vérifiable en pratique
Give product to customers early	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Fournir le produit aux clients au plus tôt est un concept du génie logiciel permettant de garder le contact avec le client. Le produit au sens large inclus le logiciel et aussi tous les produits intermédiaires du processus. 5. Formulation vérifiable en pratique

Tableau XCIII (suite)

Principes	**Commentaires sur les critères**
Communicate with customers/users	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Communication avec le client est un concept du génie logiciel. Ce concept se retrouve, entres autres, au niveau des exigences logicielles, du design, des tests, de la maintenance et de la gestion de projet. 5. Formulation vérifiable en pratique
Align incentives for developer and customer	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Developpeur et client sont des concepts du génie logiciel. De plus, Davis souligne que les objectifs des clients doivent converger et que la gestion de projet doit mettre en place des mesures pour encourager les développeurs à atteindre ces objectifs 5. Formulation vérifiable en pratique
Grow systems incrementally	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Concept du génie logiciel signalé dans les particularités des projets de développement logiciel. 5. Formulation vérifiable en pratique
Build software so that it needs a short user manual	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Manuel d'utilisation est un concept relié à la documentation du logiciel. 5. Formulation vérifiable en pratique

Tableau XCIII (suite)

Principes	**Commentaires sur les critères**
Give software tools to good engineers	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Outils CASE sont des concepts du génie logiciel 5. Formulation vérifiable en pratique
Use documentation standards	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. les normes de documentation sont des concepts du génie logiciel 5. Formulation vérifiable en pratique

Nous observons également, que deux propositions non-retenues (#15 et 16 codés en bleu) sont identifiées comme des lois candidates du génie logiciel. À cette étape, nous ne procédons qu'à leur identification.

4.4.8.2 Davis : Principes des exigences logicielles

Davis propose 23 principes pour ce thème, mais seulement deux principes satisfont aux cinq critères d'identification. Le tableau XCIV présente les résultats de l'analyse.

Tableau XCIV

Synthèse de l'analyse des principes des exigences logicielles de Davis (1995)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
38	Poor requirements yield poor cost estimates		X	X	X	X	NON
39	Determine the problem before writing requirements	X		X	X	X	NON

Tableau XCIV (suite)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
40	Determine requirements now	X	X	X	X	X	OUI
41	Fix requirements specification error now	X	X	X	X	X	OUI
42	Prototypes reduce risk in selecting user interfaces			X	X	X	NON
43	Record why requirement were included	X		X	X	X	NON
44	Identify subsets	X		X	X	X	NON
45	Review the requirements	X		X	X	X	NON
46	Avoid design in requirements	X		X	X	X	NON
47	Use the right techniques	X	X	X	X		NON
48	Use multiple views of requirements	X		X	X	X	NON
49	Organize requirements sensibly	X		X	X		NON
50	Prioritize requirements	X		X	X	X	NON
51	Write concisely	X		X		X	NON
52	Separately number every requirements	X		X	X	X	NON
53	Reduce ambiguity in requirements	X	X	X	X		NON
54	Augment, never replace, natural language	X		X		X	NON
55	Write natural language before a more formal model	X		X		X	NON
56	Keep the requirements specification readable	X		X	X		NON
57	Specify reliability specifically	X	X	X		X	NON
58	Specify when environment violates acceptable behavior	X		X	X	X	NON
59	Self-destruct TBDs			X		X	NON
60	Store requirements in a database	X		X	X	X	NON

Constat

Seulement trois propositions ne sont pas formulées de façon prescriptive. Également, 14 propositions ne satisfont pas au critère no. 2 puisque celles-ci sont soient des activités

prévues du génie logiciel ou des techniques de rédaction d'un document d'exigences logicielles. Aucune forme de dosage ou de compromis n'a été relevée parmi les 23 principes proposés. Également, cinq propositions ne comportent pas de concept explicite du génie logiciel. Finalement, quatre propositions sont identifiées comme difficilement testables et vérifiables compte tenu de l'identification problématique de la variable dépendante. Le tableau XCV présente les détails des principes retenus.

Tableau XCV

Principe des exigences logicielles retenus de Davis

Principes	**Commentaires sur les critères**
Determine requirements now	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. « Requirement » est un concept du génie logiciel. 5. Formulation vérifiable en pratique
Fix requirements specification error now	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. « Requirement specification » est un concept du génie logiciel 5. Formulation vérifiable en pratique

4.4.8.3 Principes du design

Ce thème est directement lié à la phase du design du logiciel. Davis propose 26 principes guidant le design du logiciel. Sur les 26 propositions formulées par Davis, quatre satisfont aux cinq critères d'identification. Le tableau XCVI présente les résultats de l'analyse.

Tableau XCVI

Synthèse de l'analyse des principes de design de Davis (1995)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
61	Transition from requirements to design is not easy		X	X	X		NON
62	Trace design to requirements	X		X	X	X	NON
63	Evaluate alternatives	X		X	X	X	NON
64	Design without documentation is not design			X	X	X	NON
65	Encapsulate	X		X		X	NON
66	Don't reinvent the wheel	X	X	X		X	NON
67	Keep it simple	X	X	X			NON
68	Avoid numerous special cases	X		X		X	NON
69	Minimize intellectual distance	X	X	X			NON
70	Keep design under intellectual control	X	X	X	X	X	OUI
71	Maintain conceptual integrity	X		X		X	NON
72	Conceptual errors are more significant than syntactic errors		X	X	X		NON
73	Use coupling and cohesion	X	X		X		NON
74	Design for change	X	X	X	X	X	OUI
75	Design for maintenance	X	X	X	X	X	OUI
76	Design for errors	X	X	X	X	X	OUI
77	Build generality into software	X	X	X	X		NON
78	Build flexibility into software	X	X	X	X		NON
79	Use efficient algorithms	X	X	X		X	NON
80	Module specifications provide all the information the user needs and nothing more			X	X	X	NON
81	Design is multidimensional		X	X	X		NON
82	Great designs come from great designers		X	X	X		NON
83	Know your application	X	X	X	X		NON
84	You can reuse without a big investment			X	X	X	
85	Garbage in, garbage out is incorrect		X	X		X	NON
86	Software reliability can be achieved through redundancy			X	X	X	NON

Constat

On constate que neuf propositions ne sont pas formulées de façon prescriptive et donc éliminées. Neuf propositions sont associées à des techniques, à des méthodes ou à des activités prévues du génie logiciel, elles ne sont pas retenues également. Une proposition comprend une forme de dosage et ne satisfait pas ainsi le critère trois. Huit propositions ne contiennent pas explicitement des concepts du génie logiciel ou du génie. Enfin, dix propositions ne peuvent être testées et vérifiées dans leurs conséquences du fait qu'une variable dépendante (les conséquences) ne peut être identifiée clairement. Une proposition a été identifiée comme une loi candidate du génie logiciel. Quatre propositions sont donc retenues et présentées au tableau XCVII.

Tableau XCVII

Principe de design retenus de Davis

Principes	**Commentaires sur les critères**
Keep design under intellectual control	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Design et control intellectuel sont des concepts du génie logiciel. 5. Formulation vérifiable en pratique
Design for change	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Design est un concept du génie logiciel. De plus, le changement associé au logiciel est une particularité. 5. Formulation vérifiable en pratique

Tableau XCVII (suite)

Principes	**Commentaires sur les critères**
Design for maintenance	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Design et maintenance sont des concepts du génie logiciel 5. Formulation vérifiable en pratique
Design for errors	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Design et maintenance est un concepts du génie logiciel 5. Formulation vérifiable en pratique

4.4.8.4 Principes de codage (programmation)

Ce thème est essentiellement relié à la phase de construction du logiciel. Davis propose 20 principes guidant la construction (la programmation) du logiciel. Sur les 20 propositions, une seule satisfait aux cinq critères d'identification. Le tableau XCVIII présente les résultats de l'analyse.

Tableau XCVIII

Synthèse de l'analyse des principes de codage de Davis (1995)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
87	Avoid Tricks	X	X	X		X	NON
88	Avoid global variables	X		X		X	NON
89	Write to read top-down	X		X		X	NON
90	Avoid side-effects	X		X			NON
91	Use meaningful names	X		X		X	NON
92	Write programs for people first	X	X	X	X	X	OUI

Tableau XCVIII (suite)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
93	Use optimal data structure	X	X	X		X	NON
94	Get it right before you make it faster	X	X	X			NON
95	Comment before you finalize your code	X		X	X	X	NON
96	Document before you start coding	X		X	X	X	NON
97	Hand execute every component	X		X	X	X	NON
98	Inspect code	X		X	X	X	NON
99	You can use unstructured languages		X	X		X	NON
100	Structured code is not necessarily good code			X	X	X	NON
101	Don't nest too deep	X		X		X	NON
102	Use appropriate Languages	X	X	X			NON
103	Programming language is not an excuse		X	X			NON
104	Language knowledge is not so important		X	X			NON
105	Format your programs	X		X	X	X	NON
106	Don't code too soon	X		X	X	X	NON

Constat

Concernant le critère no. 1, quatre propositions ne sont pas formulées de façon prescriptive. Douze propositions ne satisfont pas le critère no. 2. Essentiellement, dix propositions sont en fait des techniques de codage, une proposition est associée à une méthode et une autre est une activité prévue du génie logiciel. Au niveau du critère no. 4, douze propositions ne comportent pas de concept explicite du génie logiciel. Finalement, cinq propositions ne peuvent être testées et vérifiées dans leurs conséquences. Nous constatons que majoritairement, les propositions de Davis pour ce thème, sont en fait des *techniques* de codage plutôt que des principes. Une seule proposition sera donc retenue, présentée au tableau XCIX.

Tableau XCIX

Principe de codage retenu

Principes	**Commentaires sur les critères**
Write programs for people first	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Programme est un concept du génie logiciel. Egalement le sens donné au principe aborde le concept de lisibilité des programmes facilitant la maintenance 5. Formulation vérifiable en pratique

4.4.8.5 Principes de test

Davis propose 20 principes pour le thème des tests du logiciel. Sur les 20 principes proposés, seules deux propositions satisfont aux cinq critères d'identification. Le tableau C présente les résultats de l'analyse.

Tableau C

Synthèse de l'analyse des principes de test de Davis (1995)

		Critères individuels d'identification					
	Principes	#1	#2	#3	#4	#5	Retenu
107	Trace tests to requirements	X		X	X	X	NON
108	Plan tests long before it is time to test	X		X	X	X	NON
109	Don't test your own software	X	X	X	X	X	OUI
110	Don't write your own test plans	X	X	X	X	X	OUI

Tableau C (suite)

		Critères individuels d'identification					
	Principes	#1	#2	#3	#4	#5	Retenu
111	Testing exposes presence of flaws		X	X	X	X	NON
112	Though copious errors guarantee worthlessness, zero error says nothing about the value of software		X	X	X		NON
113	A successful test finds an error		X	X	X	X	NON
114	Half the errors found in 15 percent of modules		X	X	X	X	
115	Use black box and white box testing	X		X	X	X	NON
116	A test case includes expected results			X	X	X	NON
117	Test invalid inputs	X		X	X	X	NON
118	Always stress test	X		X	X	X	NON
119	The big bang theory does not apply		X	X		X	NON
120	Use McCabe complexity measure	X		X	X	X	NON
121	Use effective test completion measures	X		X	X	X	NON
122	Achieve effective test coverage	X		X	X		NON
123	Don't integrate before unit testing	X		X	X	X	NON
124	Instrument your software	X		X	X	X	NON
125	Analyze Causes for errors	X		X	X	X	NON
126	Don't take errors personally	X	X	X			NON

Constat

Au niveau du critère no. 1, six propositions ne sont pas formulées de façon prescriptive et donc éliminées. Douze propositions ne satisfont pas le critère no. 2 étant soit des techniques de test ou des activités prévues du génie logiciel. Aucune proposition ne comporte une forme de dosage ou de compromis. Deux propositions ne comportent pas de concept explicite du génie logiciel et trois propositions sont difficilement testables et

vérifiables dans leurs conséquences. De l'analyse, deux propositions seront retenues. Nous observons qu'une proposition non-retenue (#114) a été identifiée comme étant une loi candidate du génie logiciel. Le tableau CI présente les principes retenus.

Tableau CI

Principes de tests retenus

Principes	**Commentaires sur les critères**
Don't test your own software	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Test et logiciel sont des concepts du génie logiciel. 5. Formulation vérifiable en pratique
Don't write your own test plans	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Plan de test est un concept du génie logiciel. De plus, le changement associé au logiciel est une particularité. 5. Formulation vérifiable en pratique

4.4.8.6 Principes de gestion

Davis propose un volumineux bloc de 46 principes pour le thème lié à la gestion de projet. Suite à l'analyse, seules deux propositions sont retenues. Le tableau CII présente les résultats de l'analyse.

Tableau CII

Synthèse de l'analyse des principes de gestion de Davis (1995)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
127	Good management is more important that good technology		X	X			NON
128	Use appropriate solutions	X	X	X			NON
129	Don't believe everything you read	X	X	X			NON
130	Understand the customer's priorities	X	X	X		X	NON
131	People are the key to success		X	X		X	NON
132	A few good people are better than many less skilled people		X	X		X	
133	Listen to your people	X	X	X		X	NON
134	Trust your people	X	X	X		X	NON
135	Expect excellence	X	X	X		X	NON
136	Communication skills are essential		X	X	X	X	NON
137	Carry the water	X	X	X			NON
138	People are motivated by different things		X	X		X	NON
139	Keep your office quiet	X	X	X		X	NON
140	People and time are not interchangeable		X	X		X	
141	There are huge differences among software engineers		X	X	X	X	NON
142	You can optimize whatever you want		X	X		X	NON
143	Collect data unobtrusively	X		X	X	X	NON
144	Cost per line of code is not useful		X	X	X		NON
145	There is no perfect way to measure productivity		X	X	X		NON
146	Tailor cost estimation methods	X	X	X	X	X	OUI
147	Don't set unrealistic deadlines	X	X	X		X	NON
148	Avoid the impossible	X	X	X			NON
149	Know before you count	X	X	X			NON
150	Collect productivity data	X		X	X	X	NON
151	Don't forget team productivity	X	X	X			NON

Tableau CII (suite)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
152	LOC/PM independent of language		X	X	X	X	NON
153	Believe the schedule	X	X	X			NON
154	A precision-crafted cost estimate is not foolproof		X	X		X	NON
155	Reassess Schedules regularly	X		X		X	NON
156	Minor underestimates are not always bad		X	X		X	NON
157	Allocate Appropriate Resources	X		X		X	NON
158	Plan a project in detail	X		X		X	NON
159	Keep your plan up-to-date	X		X		X	NON
160	Avoid standing waves	X	X	X			NON
161	Know the top 10 risks	X		X		X	NON
162	Understand risks up front	X		X		X	NON
163	Use an appropriate process model	X		X	X	X	NON
164	The method won't save you		X	X	X	X	NON
165	No secrets for miraculous productivity increases		X	X			NON
166	Know what progress means	X	X	X		X	NON
167	Manage by variance	X		X		X	NON
168	Don't overstrain your hardware	X	X	X	X	X	OUI
169	Be optimistic about software evolution	X	X	X	X		NON
170	Be pessimistic about software evolution	X	X	X	X		NON
171	The thought that disaster is impossible often leads to disaster		X	X		X	NON
172	Do a project postmortem	X		X	X	X	NON

Constat

Nous observons que seize propositions ne sont pas formulées de façon prescriptive et ne satisfont donc la le critère no. 1. Onze propositions ne satisfont pas le critère no. 2, dont dix du fait qu'elles représentent des activités prévues du génie logiciel (ISO/IEC 12207). Aucune forme de dosage ou de compris n'est constatée dans les 46 propositions.

Cependant, 37 propositions ne contiennent pas de concept explicite au génie logiciel. À ce sujet, le génie logiciel partage une frontière avec le domaine de la gestion de projet. Au niveau du chapitre antérieur sur le génie logiciel et ses concepts, nous avons souligné que seuls les aspects particuliers à la gestion de projet logiciel seront retenus comme des concepts du génie logiciel. Ainsi, les concepts généraux de la gestion de projet ne sont pas retenus pour ce thème. Ce choix est à la base des nombreux rejets de propositions au niveau du critère no. 4. Finalement, 14 propositions ont été jugées difficilement testables et vérifiables dans leurs conséquences. Ainsi, seulement deux propositions seront donc retenues tel que présenté au tableau CIII.

Tableau CIII

Principes retenus de gestion de projet

Principes	**Commentaires sur les critères**
Tailor cost estimation methods	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Utilisation de méthode d'estimation basée sur des données provenant de la mesure 5. Formulation vérifiable en pratique
Don't overstrain your hardware	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Les limites du matériel : concept que le génie logiciel doit tenir compte. 5. Formulation vérifiable en pratique

4.4.8.7 Principes d'assurance qualité

Davis regroupe sous ce thème douze principes liés à la gestion des configurations, à l'assurance qualité et à la vérification et validation. Suite à l'analyse, une seule proposition est retenue. Le tableau CIV présente les résultats.

Tableau CIV

Synthèse de l'analyse des principes d'assurance qualité de Davis (1995)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
173	Product assurance is not a luxury		X	X	X		NON
174	Establish SCM procedure early	X		X	X	X	NON
175	Adapt SCM to software process	X		X	X	X	NON
176	Organize SCM to be independent of project management	X		X	X	X	NON
177	Rotate people through product assurance	X	X	X	X	X	OUI
178	Give every intermediate product a name and version	X		X	X	X	NON
179	Control baselines	X		X		X	NON
180	Save everything	X		X		X	NON
181	Keep track of every change	X		X	X	X	NON
182	Don't bypass change control	X		X	X	X	NON
183	Rank and schedule change requests	X		X	X	X	NON
184	Use validation and verification on large developments			X	X	X	NON

Constat

Deux propositions ne sont pas formulées d'une façon prescriptive et ne satisfont donc pas le critère no. 1. Au niveau du critère no. 2, dix propositions sont rejetées du fait qu'elles représentent des activités prévues du génie logiciel (ISO/IEC 12207). Aucune proposition ne comporte une forme de compromis ou de dosage entre des concepts. Au niveau du critère no. 4, deux propositions ne comportent pas concept explicite du génie logiciel. Au niveau du critère no. 5, une seule proposition ne peut être testée et vérifiée dans ses conséquences. Une seule proposition est donc retenue de ce thème et présentée au tableau CV.

Tableau CV

Principes d'assurance qualité retenus (Davis 1995)

Principes	**Commentaires sur les critères**
Rotate people through product assurance	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Assurance qualité du logiciel est un concept du génie logiciel 5. Formulation vérifiable en pratique

4.4.8.8 Principes d'évolution (maintenance)

Davis regroupe sous ce thème 17 principes associés à l'évolution du logiciel. L'auteur souligne que l'évolution comprend une suite d'activités pour ajouter de nouvelles fonctions au logiciel, pour optimiser le fonctionnement du logiciel ou pour corriger des problèmes. Le thème de l'évolution est associé à la phase de maintenance du logiciel. Parmi les 17 propositions faites par Davis, aucune ne satisfait aux critères d'identification de l'analyse. Le tableau CVI présente les résultats de l'analyse.

Tableau CVI

Synthèse de l'analyse des principes d'évolution de Davis (1995)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
185	Software will continue to change		X	X	X	X	
186	Software's entropy increases		X	X	X	X	
187	If it isn't broke, don't fix it.	X	X	X		X	NON
188	Fix problems, not symptoms	X	X	X		X	NON
189	Change requirements first	X		X	X	X	NON
190	Prerelease errors yield post release errors		X	X	X	X	NON
191	The older a program, the more difficult it is to maintain		X	X	X	X	

Tableau CVI (suite)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
192	Language affects maintainability		X	X	X	X	
193	Sometimes it is better to start over		X	X		X	NON
194	Renovate the worst first	X	X	X		X	NON
195	Maintenance causes more errors than development		X	X	X	X	
196	Regression test after every change	X		X	X	X	NON
197	Belief that a change is easy makes it likely it will be made incorrectly		X	X		X	NON
198	Structuring unstructured code does not necessarily improve it		X	X	X	X	NON
199	Use profiler before optimizing	X		X		X	NON
200	Conserve familiarity	X	X	X			NON
201	The system's existence promotes evolution		X	X	X	X	NON

Constat

Dix propositions ne satisfont pas le critère no. 1 n'étant pas formulées d'une façon prescriptive. Trois propositions ne satisfont pas le critère no. 2, deux découlant d'une méthode et l'autre d'une technique d'optimisation. Aucune proposition ne comporte une forme de dosage ou de compromis. Sept propositions ne comportent pas de concept explicite du génie logiciel, certaines comportant des concepts du domaine de l'informatique. Tous les principes proposés peuvent être testés et vérifiés dans leurs conséquences. Aucun principe proposé n'est retenue pour ce thème. Cependant, cinq propositions sont identifiées comme étant des lois potentielles du génie logiciel.

4.4.8.9 Synthèse sur les propositions de Davis

Davis a proposé un volumineux bloc de 201 principes. Suite à l'analyse faite des ces principes en considérant les cinq critères d'identification individuel, seules 20 propositions sont retenues pour la suite. On constate que 66 principes ne sont pas formulés de façon prescriptive et 88 principes représentent des activités prévues du génie logiciel ou des techniques ou des parties de méthodes. Seuls trois principes comportent une forme de dosage ou de compromis dans leur formulation. Fait intéressant à souligner, 72 principes ne comportent pas de concept du génie logiciel ou même du génie. Le thème des principes de gestion obtient un ratio élevé pour ce critère, compte tenu que les principes comportent beaucoup de concepts provenant d'une discipline connexe qui est la gestion de projet. Finalement, 54 principes ne peuvent être testés dans leurs conséquences, faute d'identifier clairement une variable dépendante. Nous observons que onze principes non retenus sont identifiés comme étant des lois candidates du génie logiciel.

4.4.9 Karl Wiegers (1996)

Wiegers (1996) identifie 14 principes qui auraient une influence sur la culture du génie logiciel et implicitement sur la qualité et l'efficacité du processus de développement. Parmi les 14 propositions faites par Wiegers, deux satisfont aux critères d'identification de l'analyse. Le tableau CVII présente les résultats de l'analyse.

Tableau CVII

Synthèse de l'analyse des principes de Wiegers (1996)

		Critères individuels d'identification					
	Principes	#1	#2	#3	#4	#5	Retenu
1	Never let your boss or your customer talk you into doing a bad job	X	X	X			NON

Tableau CVII (suite)

		Critères individuels d'identification					
	Principes	#1	#2	#3	#4	#5	Retenu
2	People need to feel the work they do is appreciated		X	X		X	NON
3	Ongoing education is every team member's responsibility		X	X		X	**NON**
4	Customer involvement is the most critical factor in software quality		X	X	X	X	**NON**
5	Your greatest challenge is sharing the vision of the final product with the customer		X	X	X		NON
6	Continual improvement of your software development process is both possible and essential	X	X	X	X		NON
7	Written software development procedures can help build a shared culture of best practices			X	X	X	NON
8	Quality is the top priority; long term productivity is a natural consequence of high quality	X	X	X	X	X	OUI
9	Strive to have a peer, rather than a customer, find a defect	X	X	X	X	X	**OUI**
10	A key to software quality is to iterate many times on all development steps except coding : do this once	X		X	X	X	NON
11	Managing bug reports and change request is essential to controlling quality and maintenance			X	X	X	NON
12	If you measure what you do, you can learn to do it better		X	X	X	X	NON
13	Do what makes sense; don't resort to dogma	X	X	X			NON
14	You can't change everything at once. Identify those changes that will yield the greatest benefits and begin to implement them next Monday	X		X	X	X	NON

Constat

Sept propositions ne sont pas formulées de façon prescriptive et ne satisfont donc pas le critère no. 1. Deux propositions (6 et 8) sont formulées indirectement de façon prescriptive en utilisant des qualificatifs tels « essential » et « top priority », on considère qu'elles satisfont au critère no. 1 puisque ces qualificatifs sont considérés comme prescriptif. Trois propositions sont en fait des activités déjà prévues à la ISO/IEC 12207 et une découle d'une méthode et technique. Aucune proposition ne contient de compromis ou de dosage entre des concepts. Cinq propositions ne contiennent pas explicitement de concepts du génie logiciel ou de concepts généraux du génie. Enfin, seulement quatre propositions ne satisfont pas au critère cinq. Ainsi, deux propositions sont donc retenues et présentées au tableau CVIII.

Tableau CVIII

Principes retenus de Wieger (1996)

Principes	**Commentaires sur les critères**
Quality is the top priority; long term productivity is a natural consequence of high quality	1. Formulation n'est pas directement prescriptive, mais l'utilisation du qualificatif « top priority » amène une forme prescriptive. 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Qualité du logiciel est un concept du génie logiciel 5. Formulation vérifiable en pratique
Strive to have a peer, rather than a customer, find a defect	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Le concept d'activités d'assurance qualité (inspection, revue technique) est du génie logiciel 5. Formulation vérifiable en pratique

4.4.10 Anthony Wasserman (1996)

Wasserman (1996) présente une liste de huit « concepts fondamentaux » du génie logiciel. L'auteur souligne que ces concepts fondamentaux sont au cœur même des meilleures pratiques de l'industrie. De ces huit propositions, aucune n'est retenue suite à l'analyse effectuée. Le tableau CIX présente la synthèse de l'analyse.

Tableau CIX

Synthèse de l'analyse des principes d'évolution de Wasserman (1996)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Abstraction (including information hiding)			X			NON
2	Analysis and design methods and notation			X	X		NON
3	User interface prototyping			X	X		NON
4	Modularity and architecture Separation of concerns Localization Decomposition Design patterns			X	X		NON
5	Software life cycle and process			X	X		NON
6	Reuse			X	X		NON
7	Metrics			X	X		NON
8	Tools and integrated environment			X	X		NON

Constat

En premier lieu, l'auteur présente des concepts et non des propositions prescriptives. Ainsi, tous les énoncés ne satisfont pas au premier critère. Les concepts présentés sont en fait des techniques ou des activités du génie logiciel; ainsi, aucun énoncé ne satisfait le deuxième critère également. Tous les énoncés satisfont le troisième critère, puisqu'aucun ne comporte de compromis ou de dosage. Un seul énoncé ne comporte pas

de concept explicite du génie logiciel ou du génie. Au niveau de cinquième critère, aucun énoncé ne peut être testé compte tenu de la généralité des concepts présentés et de la difficulté d'identifier une variable dépendante. Aucune proposition n'est donc retenue.

4.4.11 Paul Taylor (2001)

Taylor (2001) utilise les principes de design de Mayall (1979) comme cadre conceptuel descriptif dans le but de vérifier si ces principes peuvent s'appliquer au génie logiciel. Mayall a établi ces principes dans le but de les présenter aux étudiants du génie. Sur les dix énoncés proposés, aucun ne sera retenu. Le tableau CX présente les résultats de l'analyse effectuée.

Tableau CX

Synthèse de l'analyse des principes d'évolution de Taylor (2001)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Totality		X	X			NON
2	Time		X	X			NON
3	Value		X	X			NON
4	Resources		X	X	X		NON
5	Synthesis		X	X	X		NON
6	Iteration		X	X	X		NON
7	Change		X	X			NON
8	Roles and Relationships		X	X			NON
9	Competence		X	X			NON
10	Service		X	X			NON

Constat

Les énoncés sont en fait des concepts et non des propositions prescriptives. Ainsi aucun énoncé ne satisfait au premier critère. Tous les énoncés satisfont aux critères deux et trois. Le critère quatre n'est pas satisfait pour sept énoncés. Les concepts sont de nature

générale et non spécifique ou particulière au génie logiciel ou génie. Également, les concepts ne peuvent être vérifiés dans leurs conséquences compte tenu du niveau général de ceux-ci et du problème à identifier les variables dépendantes. Aucune proposition ne sera donc retenue.

4.4.12 Bertrand Meyer (2001)

Meyer (2001) présente treize principes ou concepts fondamentaux qui représentent les fondements de la connaissance qu'un professionnel en logiciel doit connaitre. Suite à l'analyse aucun énoncé n'est retenu. Le tableau CXI présente la synthèse de l'analyse effectuée.

Tableau CXI

Synthèse de l'analyse des principes d'évolution de Meyer (2001)

		Critères individuels d'identification					
	Principes	#1	#2	#3	#4	#5	Retenu
1	Abstraction		X	X			NON
2	Distinction between specification and implementation		X	X	X		NON
3	Recursion			X			NON
4	Information hiding			X			NON
5	Reuse		X	X	X		NON
6	Battling complexity		X	X	X		NON
7	Scaling up		X	X	X		NON
8	Designing for change		X	X	X	X	NON
9	Classification			X			NON
10	Typing			X			NON
11	Contracts			X			NON
12	Exception handling			X	X		NON
13	Errors and debugging			X	X		NON

Constat

Aucun énoncé n'est formulé comme une proposition prescriptive, la majorité étant des concepts. Ainsi, aucun ne satisfait donc au premier critère. Au niveau du critère deux, sept énoncés ne sont pas retenus du fait qu'ils représentent des techniques associées à un langage de programmation. Tous les énoncés satisfont au critère trois, aucun ne comportant un compromis ou une forme de dosage. Concernant le critère quatre, six énoncés ne comportent pas de concepts explicite du génie logiciel. De ceux-ci, cinq sont directement liés à des concepts de l'informatique. Douze énoncés ne peuvent être testés et vérifiés dans leurs conséquences compte tenu qu'il n'est pas possible d'identifier les conséquences. Ainsi, aucune des propositions ne sera donc retenue.

4.4.13 Bourque, Dupuis, Abran, Moore, Tripp et Wolf (2002)

Bourque et al. (2002) ont conduit une recherche empirique auprès de plus de 600 personnes considérées comme des experts du domaine du génie logiciel. La méthode de recherche utilisée a permis d'identifier plusieurs principes candidats. Les auteurs ont par la suite entrepris une importante synthèse qui a mené à l'identification de 15 principes candidats. Suite à l'analyse effectuée, dix principes ont satisfait aux cinq critères. Le tableau CXII présente la synthèse de l'analyse.

Tableau CXII

Synthèse de l'analyse des principes candidats de Bourque et al. (2001)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
1	Apply and use quantitative measurements in decision making	X	X	X	X	X	OUI
2	Build with and for reuse	X	X	X	X	X	OUI
3	Control complexity with multiple perspectives and multiple levels of abstraction	X	X	X	X		NON

Tableau CXII (suite)

		Critères individuels d'identification					
	Principes	**#1**	**#2**	**#3**	**#4**	**#5**	**Retenu**
4	Define software artifacts rigorously	X	X	X	X	X	OUI
5	Establish a software process that provides flexibility	X	X	X	X	X	OUI
6	Implement a disciplined approach and improve it continuously	X	X	X	X	X	OUI
7	Invest in the understanding of the problem	X	X	X	X	X	OUI
8	Manage quality throughout the life cycle as formally as possible	X	X	X	X		NON
9	Minimize software component interaction	X		X	X	X	NON
10	Produce software in a stepwise fashion	X	X	X	X	X	OUI
11	Set quality objectives for each deliverable product	X		X	X	X	NON
12	Since change is inherent to software, plan for it and manage it	X	X	X	X	X	OUI
13	Since tradeoffs are inherent to software engineering, make them explicit and document it	X	X	X	X	X	OUI
14	To improve design, study previous solutions to similar problems	X	X	X	X	X	OUI
15	Uncertainty is unavoidable in software engineering. Identify and manage it	X		X	X	X	NON

Constat

Une première observation importante, à l'effet que toutes les propositions sont formulées de façon prescriptive et satisfont ainsi, le critère no. 1. Nous soulignons que c'est le seul groupe d'auteurs qui atteint un score parfait pour ce critère. Douze propositions satisfont au critère no. 2. Les trois propositions non retenues, sont en fait des activités prévues à la norme ISO/IEC 12207. Toutes les propositions satisfont le

critère no. 3 concernant les compromis et le dosage. À ce propos, le contraire aurait été étonnant, compte tenu que les auteurs ont proposé ce critère qui a été retenu pour notre analyse. Toutes les propositions satisfont le critère no. 4. Concernant le critère no. 5, deux formulations ne permettent pas d'identifier clairement une variable dépendante permettant d'observer des conséquences de l'application des ces principes. Dix propositions sont donc retenues et présentées au tableau CXIII.

Tableau CXIII

Principes retenus de Bourque et al. (2002)

Principes	**Commentaires sur les critères**
Apply and use quantitative measurements in decision making	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Application et l'utilisation de mesures sont des concepts mentionnés au chapitre du « software engineering management » de SWEBOK 5. Formulation vérifiable en pratique
Build with and for reuse	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. La réutilisation est un concept du génie logiciel 5. Formulation vérifiable en pratique
Define software artifacts rigorously	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Le « software artifact » est un concept du génie logiciel 5. Formulation vérifiable en pratique
Establish a software process that provides flexibility	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. « software process » est un concept du génie logiciel 5. Formulation vérifiable en pratique

Tableau CXIII (suite)

Principes	**Commentaires sur les critères**
Implement a disciplined approach and improve it continuously	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Une approche disciplinée est un composant de la définition donnée par SWEBOK pour le génie logiciel 5. Formulation vérifiable en pratique
Invest in the understanding of the problem	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. La compréhension du problème n'est pas un concept spécifique au génie logiciel, mais un concept important du processus d'ingénierie 5. Formulation vérifiable en pratique
Minimize software component interaction	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Composant logiciel et l'interaction entre eux sont des concepts du génie logiciel 5. Formulation vérifiable en pratique
Since change is inherent to software, plan for it and manage it	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Le changement est une particularité qui a un impact important en génie logiciel. Ainsi, il est considéré comme un concept du génie logiciel 5. Formulation vérifiable en pratique
Since tradeoffs are inherent to software engineering, make them explicit and document it	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Compromis : concept du génie logiciel et du génie 5. Formulation vérifiable en pratique

Tableau CXIII (suite)

Principes	**Commentaires sur les critères**
To improve design, study previous solutions to similar problems	1. Formulation prescriptive 2. Ne découle pas d'une activité, méthode ou d'une technique 3. Aucun compromis 4. Le design est un concept du génie logiciel. Il y a aussi une notion de réutilisation dans la formulation 5. Formulation vérifiable en pratique

4.4.14 Ghezzi, Jazayeri et Mandrioli (2003)

Ghezzi et al. (2003) ont publié un livre sur les fondements du génie logiciel qui a la particularité de présenter au lecteur l'application des principes du génie logiciel aux différentes phases du processus de développement. Suite à l'analyse, aucun des énoncés proposés n'est retenu. Le tableau CXIV présente la synthèse de l'analyse effectuée.

Tableau CXIV

Synthèse de l'analyse des principes d'évolution de Ghezzi et al. (2003)

		Critères d'identification					
	Principes	#1	#2	#3	#4	#5	Retenu
1	Rigor and formality		X	X			NON
2	Separation of concerns		X	X		X	NON
3	Modularity			X			NON
4	Abstraction			X			NON
5	Anticipation of change		X	X			NON
6	Generality		X	X			NON
7	Incrementality			X	X		NON

Constat

En premier lieu, aucun des énoncés n'est formulé comme une proposition prescriptive; ainsi, aucun ne satisfait donc le premier critère. Trois énoncés ne satisfont pas le critère

no. 2, puisqu'ils sont soit associés à des techniques ou à une méthode. Aucun énoncé ne manifeste une forme de compromis ou de dosage. Au niveau du critère no. 4, cinq des sept énoncés ne contiennent pas de concept explicite du génie logiciel ou du génie. Compte tenu que les énoncés sont composés d'un ou deux mots, six énoncés ne peuvent être testés et vérifiés dans leurs conséquences dues à la difficulté d'identifier la variable dépendante. Ainsi, aucune proposition n'est donc retenue.

4.5 Synthèse de la phase 2

La deuxième phase a permis d'analyser 308 principes. L'application des cinq critères individuels d'identification a permis de filtrer les 308 principes pour n'en conserver que 35 propositions. Ces 35 propositions satisfont aux cinq critères individuels. Le tableau CXV présente les principes retenus.

Tableau CXV

Principes satisfaisant aux cinq critères individuels

1. Align incentives for developer and customer	19. Grow systems incrementally
2. Apply and use quantitative measurements in decision making	20. Implement a disciplined approach and improve it continuously
3. Build software so that it needs a short user manual	21. Invest in the understanding of the problem
4. Build with and for reuse	22. Involve the customer
5. Communicate with customers/users	23. Keep design under intellectual control
6. Define software artifacts rigorously	24. Maintain clear accountability for results
7. Design for change	25. Produce software in a stepwise fashion
8. Design for errors	26. Quality is the top priority; long term productivity is a natural consequence of high quality
9. Design for maintenance	27. Rotate people through product assurance

Tableau CXV (suite)

10. Determine requirements now	28. Since change is inherent to software, plan for it and manage it
11. Don't overstrain your hardware	29. Since tradeoffs are inherent to software engineering, make them explicit and document it
12. Don't test your own software	30. Strive to have a peer, rather than a customer, find a defect
13. Don't try to retrofit quality	31. Tailor cost estimation methods
14. Don't write your own test plans	32. To improve design, study previous solutions to similar problems
15. Establish a software process that provides flexibility	33. Use better and fewer people
16. Fix requirements specification error now	34. Use documentation standards
17. Give product to customers early	35. Write programs for people first
18. Give software tools to good engineers	

Globalement, quatre des cinq critères ont éliminé presqu'un nombre équivalent de principes (122 à 137). Bien entendu, il y des principes communs à plus d'un critère qui ont été éliminés. Le critère no. 3 n'a été utilisé pour éliminer que trois principes. Les trois principes en question ont tous été proposés par Davis (1995). Ainsi, nous observons que 305 propositions ne comportent pas explicitement un dosage ou une forme de compromis dans leur formulation.

Nous observons que 129 propositions ne sont pas formulées de façon prescriptive, telle une règle d'action. De ce nombre, 52 ne sont qu'un concept tel « abstraction », ce qui ne peut satisfaire la définition donnée au terme principe à l'effet qu'un principe doit être une proposition prescriptive.

Le critère no. 2 a servi essentiellement à identifier les principes qui étaient en fait des activités répertoriées à la norme ISO/IEC 12207 du génie logiciel. Un principe n'est pas en soi une activité, mais des activités peuvent cependant en découler. Également, le

critère no. 2 a permis d'identifier les principes qui étaient plutôt des techniques ou des étapes d'une méthode.

Le critère no. 4 a permis de retrancher 126 principes qui ne comportaient pas explicitement de concepts du génie logiciel. Au second tour, on a élargi la portée de ce critère pour englober les concepts reliés au génie. Ainsi, si le principe ne comporte pas explicitement un concept du génie logiciel, mais un concept du génie, le critère est satisfait. Cependant, les concepts en provenance de disciplines limitrophes telles l'informatique et la gestion de projet n'ont pas été retenus lors de l'analyse. Ainsi, les principes comportant des concepts de l'informatique ou de la gestion de projet, par exemple, sont tous potentiellement des principes de ces disciplines et non du génie logiciel.

L'application du critère no. 5 a été particulièrement complexe. Dans la majorité des cas, la formulation du principe n'inclut pas les conséquences de l'appliquer ou non. Ainsi, il a été nécessaire de consulter l'explication donnée par l'auteur, lorsque disponible, afin d'évaluer les conséquences possibles. En dernier lieu, si l'explication ne permettait toujours pas d'identifier les conséquences, nous avons évalué si une expérimentation pouvait être faite. Ainsi, un projet À met en pratique le principe et un projet B ne l'applique pas. Si c'est possible d'observer des différences notables, alors le principe satisfait au critère no. 5. Certaines formulations ne permettent pas d'identifier clairement des conséquences. À titre d'exemple, le principe « Case tool are expensive » ne représente pas une formulation vérifiable dans ses conséquences sur le logiciel. Seulement 12 principes ont été écartés sur ce critère alors qu'ils satisfaisaient aux quatre autres critères. À titre d'exemple, le principe « Built flexibilty into software » ne satisfait pas au critère no. 5. Qu'est ce que la flexibilité dans le logiciel ? Comment l'identifier ? Ainsi, dans ces cas, la variable dépendante (les conséquences) ne peut être identifiée précisément.

Le tableau CXVI présente pour chacun des critères le nombre de principes retenus ou non.

Tableau CXVI

Synthèse de l'impact de l'application des critères individuels

	Conservés	**Non retenus**
Critère no.1	179	129
Critère no.2	175	133
Critère no.3	305	3
Critère no.4	182	126
Critère no.5	186	122

4.6 Le cas des principes éliminés sur le critère 1

À la lumière des résultats de la phase 2, nous constatons que 18 principes (tableau CXVII) ont été écartés seulement sur le critère no. 1, tout en satisfaisant aux quatre autres critères. Cette constatation nous amène à poser les questions de recherche suivantes : est-ce que des propositions intéressantes auraient été écartées seulement à cause de leur formulation non prescriptive? Est-ce qu'une reformulation mineure de la proposition permettrait de les récupérer?

Dans cette section, chacune des 18 propositions écartées est revue afin de vérifier si une reformulation est possible. La reformulation doit être mineure et non exiger la réécriture complète de la proposition. L'objectif de cette recherche n'est pas de créer des principes du génie logiciel, mais de mieux les identifier parmi les centaines de propositions proposées par la littérature.

Tableau CXVII

Principes rejetés sur le critère no. 1

1. Productivity and Quality are inseparable 2. Technique before tools 3. Poor requirements yield poor cost estimates 4. Testing exposes presence of flaw 5. A successful test finds an error 6. Half the errors found in 15 percent of modules 7. LOC/PM independent of language 8. The method won't save you 9. Software will continue to change 10. Software's entropy increases	11. Prerelease errors yield post release errors 12. The older a program, the more difficult it is to maintain 13. Language affect maintainability 14. Maintenance causes more errors than development 15. Structuring unstructured code does not necessarily improve it 16. The system's existence promotes evolution 17. Customer involvement is the most critical factor in software quality 18. Designing for change

Pour évaluer la pertinence d'une reformulation d'une proposition, nous retournerons à l'explication même fournie par les auteurs pour chacune des propositions afin de bien saisir le sens donné à la proposition. L'évaluation doit vérifier si l'explication de l'auteur guide l'action; le cas échéant, une reformulation sera tentée.

Tableau CXVIII

Processus de réévaluation

Intrants	**Processus**	**Extrants**
▪ 18 propositions écartées sur le critère no.1 ▪ Explications des auteurs	▪ Le sens guide-t-il l'action? ▪ Est-ce reformulable?	▪ Propositions reformulées ▪ Propositions toujours écartées ▪ Fiches d'évaluation

Pour documenter le processus, nous utiliserons des fiches individuelles, tout comme lors de l'analyse de l'ensemble des principes. Nous procéderons à l'aide d'une fiche

individuelle pour chacun des principes. Sur la fiche se retrouve l'explication donnée par l'auteur sur la proposition et un tableau indiquant les possibilités de reformulation et enfin le résultat. Les 18 fiches d'évaluation se retrouvent à l'annexe 3. Cependant, les grandes lignes sont présentées dans cette section.

1. "Productivity and Quality are inseparable" (Davis 1995)

   L'auteur souligne que la qualité et la productivité sont inséparables. Ainsi, plus les attentes de qualité sont élevées, moins la productivité le sera. À l'inverse, moins il y a d'exigences de qualité, plus la productivité serait grande. La proposition, ainsi que l'explication de l'auteur confirment le sens descriptif et factuel de la proposition qui ne mène pas à l'action. Cette proposition ne se prête donc pas à une reformulation et elle ne sera pas retenue.

2. "Technique before tools" (Davis 1995)

   L'auteur souligne que l'ingénieur logiciel doit maîtriser les techniques du génie logiciel avant d'être en mesure de bien utiliser les outils de développement. Le sens donné par l'auteur guide une forme d'action. Une première reformulation pourrait être : « Put technique before tools ». Même si la proposition est maintenant prescriptive, elle ne guide pas précisément l'action. En reprenant les termes utilisés par l'auteur dans l'explication, une seconde reformulation est suggérée : « Know software engineering's techniques before using development tools ». Cette formulation précise mieux le sens de l'auteur tout en indiquant une action. Ainsi, la seconde reformulation sera donc retenue.

3. "Poor requirements yield poor cost estimates" (Davis 1995)

   L'auteur présente cinq causes majeures d'une mauvaise estimation, causes directement liées à la phase des exigences logicielles. Dans ce cas-ci, le sens du principe est clairement descriptif et factuel. Les actions à prendre ne sont pas mentionnées, tant au niveau de la formulation du principe qu'au niveau de

l'explication de l'auteur. Cette proposition ne se prête à une reformulation mineure et elle n'est donc pas retenue.

4. "Testing exposes presence of flaw" (Davis 1995)

   L'auteur souligne que les tests font ressortir la présence d'erreur, mais ne confirment rien de leur absence. Les tests peuvent augmenter le niveau de confiance à l'effet que le logiciel se comporte correctement, mais ils ne prouvent pas que le logiciel soit sans défaut. La proposition et l'explication de l'auteur confirme la nature descriptive et factuelle. Ainsi, cette proposition ne se prête pas à une reformulation mineure et elle n'est donc pas retenue.

5. "A successful test finds an error" (Davis 1995)

   L'auteur souligne qu'un bon test doit permettre de détecter des erreurs. De plus, l'auteur identifie une action à faire à l'effet de choisir des cas de tests dont la probabilité qu'ils détectent des erreurs est élevée. Une reformulation peut être faite en reprenant les propos mêmes de son explication : « Select tests based on the likelihood that they will find faults ». Cette reformulation guide l'action à entreprendre et elle sera donc retenue.

6. "Half the errors found in 15 percent of modules"

   L'auteur souligne que la moitié des erreurs sont détectées dans seulement 15% des modules. L'explication de l'auteur confirme la nature descriptive et factuelle de la proposition. Ainsi, la proposition ne se prête pas à une reformulation mineure guidant l'action. La proposition n'est pas retenue.

7. "LOC/PM independent of language"

   L'auteur souligne qu'il existe une croyance générale à l'effet que la quantité de lignes de codes (LOC) écrites par un programmeur est indépendante du langage de programmation. À l'opposé, il souligne également, que cette croyance ne serait pas

fondée et qu'au contraire, il serait requis de connaître le langage de programmation qui sera utilisé afin de mieux estimer l'effort de programmation. Nous constatons que l'explication de l'auteur diverge de la proposition. Ainsi, elle ne se prête pas à une reformulation mineure. La proposition n'est donc pas retenue.

8. "The method won't save you" (Davis 1995)

   L'auteur affirme que la croyance envers les méthodes de développement est exagérée. Les méthodes sont utiles et efficaces que si elles sont intégrées dans un processus rigoureux, organisé et planifié. Les entreprises qui peu de rigueur dans le processus de développement vont rester médiocre même en choisissant les meilleures méthodes. Le sens de la proposition est essentiellement descriptif et ne guide pas l'action à entreprendre, ce que confirme l'explication fournie par l'auteur. Cette proposition ne se prête pas à une reformulation mineure. Elle n'est donc pas retenue.

9. "Software will continue to change" (Davis 1995)

   L'auteur explique qu'un système logiciel subira des modifications continuelles pour suivre les changements associés au monde réel. Le logiciel va continuer à être modifié jusqu'au moment où il sera plus avantageux de le remplacer. L'explication fournit par l'auteur confirme la nature descriptive de la proposition qui ne se prête donc pas à une reformulation mineure. Elle n'est donc pas retenue.

10. "Software's entropy increases" (Davis 1995)

    L'auteur affirme qu'un logiciel qui subit des modifications deviendra de plus en plus complexe et de moins en moins structuré. L'explication de l'auteur confirme le sens descriptif de la proposition. De plus, celle-ci ne guide pas l'action. Ainsi, la proposition ne se prête pas à une reformulation mineure et elle n'est donc pas retenue.

11. "Prerelease errors yield post release errors" (Davis 1995)

    L'auteur affirme qu'un programme ou un composant logiciel qui a connu un grand nombre de défauts en cours de développement, comportera un grand nombre de défauts en production. L'auteur suggère, dans ce cas, de réécrire le composant en question. La proposition est descriptive et factuelle. Celle-ci ne guide pas l'action à prendre. Malgré que l'auteur suggère une action à prendre, il faudrait écrire une toute nouvelle proposition pour remplacer l'existante. Comme nous avons fixé la limite à une reformulation mineure, cette proposition ne peut être retenue.

12. "The older a program, the more difficult it is to maintain" (Davis 1995)

    L'auteur affirme qu'un logiciel vieillit, le nombre de modules qui doivent être modifiés pour intégrer une modification augmente. De plus, au fil des changements, la structure du logiciel se dégrade rendant encore plus difficile les changements ultérieurs. L'explication de l'auteur confirme la nature descriptive et factuelle de la proposition. Celle-ci ne guide pas l'action à entreprendre. Ainsi, la proposition ne se prête pas à une reformulation et elle n'est pas donc retenue.

13. "Language affects maintainability" (Davis 1995)

    L'auteur souligne que le choix du langage de programmation affecte directement l'effort de maintenance. Les langages qui forcent ou qui facilitent une forte cohésion et un faible couplage permettent de faciliter le développement et de diminuer l'effort de maintenance. Une reformulation possible serait : « Choose a programming language to improve maintainability ». La reformulation mineure proposée guide l'action et représente mieux le sens donné par l'explication de l'auteur. La reformulation est donc retenue.

14. "Maintenance causes more errors than development" (Davis 1995)

    L'auteur affirme que la maintenance provoque plus d'erreurs que le développement initial du logiciel. L'explication de l'auteur confirme la nature descriptive de la

proposition et celle-ci ne guide aucunement l'action à entreprendre. La proposition ne se prête pas à une reformulation mineure et elle n'est donc pas retenue.

15. "Structuring unstructured code does not necessarily improve it" (Davis 1995)

    L'auteur souligne qu'en présence d'un programme dont le code n'est pas structuré, il ne serait pas souhaitable de tenter de le restructurer. L'auteur affirme que restructurer le programme n'améliora pas la qualité du code dans l'ensemble. Ainsi, il est suggéré de repenser et de faire une nouvelle conception du programme. L'auteur identifie clairement l'action à entreprendre dans l'explication. Ainsi, il est possible de reformuler la proposition comme suit : «In face of unstructured code, rethink the module and redesign it from scratch. ». La reformulation est un peu plus que mineure, mais elle reprend les propos même de l'auteur. La reformulation guide l'action et elle sera retenue.

16. "The system's existence promotes evolution" (Davis 1995)

    L'auteur souligne que même s'il n'y aucun changement aux spécifications durant le développement, l'environnement du système continue à évoluer et d'une façon incontournable, il y aura des changements à effectuer au logiciel. Ainsi, l'auteur suggère de planifier les changements à faire après le déploiement du logiciel. La proposition de l'auteur ne guide pas l'action suggérée. Il faudrait réécrire entièrement la proposition pour qu'elle tienne compte de l'action suggérée par l'explication de l'auteur Ainsi, elle ne se prête pas à une reformulation mineure et elle n'est donc pas retenue.

17. "Customer involvement is the most critical factor in software quality" (Wiegers 1996)

    L'auteur souligne que l'implication du client est nécessaire et incontournable principalement au niveau de la définition des exigences du logiciel. Une reformulation mineure est possible : « Involve the customer ». Cependant, la

reformulation nous offre une proposition prescriptive similaire à celle déjà retenue de Royce (1970). La reformulation n'enrichit pas notre bassin de principes. La proposition ne sera donc pas retenue.

18. "Designing for change"

    Cette proposition peut facilement être reformulée de façon prescriptive comme suit : « Design for change ». Cependant, la reformulation est identique à une autre proposition déjà retenue, soit la proposition no. 7. La reformulation n'ajoute qu'une duplication non utile. Ainsi, la proposition n'est donc pas retenue.

Le processus de réévaluation a revu 18 propositions écartées sur la non satisfaction du critère no. 1. Le processus a permis la reformulation mineure de quatre propositions tel que présenté au tableau CXIX. Les propositions reformulées seront intégrées aux autres propositions retenues de la phase2.

Tableau CXIX

Propositions reformulée et retenues

Formulation originale	**Reformulation proposée**
1. Technique before tools	Know software engineering's techniques before using development tools
2. A successful test finds an error	Select tests based on the likelihood that they will find faults
3. Language affect maintainability	Choose a programming language according to maintainability
4. Structuring unstructured code does not necessarily improve it	In face of unstructured code, rethink the module and redesign it from scratch.

Le tableau CXX présente une ventilation par auteurs du nombre de principes proposés, du nombre retenu et du nombre de lois empiriques potentielles identifiées.

Tableau CXX

Sommaire des principes proposés et retenus par auteur

Auteurs	Principes proposés	Principes retenus	Lois candidates
Winston W. Royce (1970)	5	1	0
Ross, Goodenough et Irvine (1975)	7	0	0
H.D. Mills (1980)	3	0	0
Many Lehman (1980)	5	0	2
Barry W. Boehm (1983)	7	2	0
Booch et Bryan (1984)	7	0	0
Buschmann et al. (1996)	11	0	0
Alan Davis (1995)	201	24	11
Karl Wiegers (1996)	14	2	0
Anthony Wasserman (1996)	8	0	0
Paul Taylor (2001)	10	0	0
Bertrand Meyer (2001)	13	0	0
Bourque, Dupuis, Abran, Moore, Tripp et Wolf (2002)	15	10	0
Ghezzi, Jazayeri et Mandrioli (2003)	7	0	0

Au niveau du chapitre antérieur sur la définition des termes concept et principe, nous avons fait mention d'une hypothèse à l'effet que certaines propositions pourraient être plutôt des lois empiriques que des principes. À titre de rappel, une loi empirique est une généralisation venant essentiellement d'observations de la pratique. Une loi serait moins générale qu'un principe et elle peut ne pas toujours être vraie, contrairement à un principe qui est une vérité fondamentale.

Le tableau CXXI présente 13 énoncés de lois empiriques potentielles que nous avons identifiées au cours de l'évaluation des principes. Nous constatons que le thème du changement se retrouve dans six des propositions. D'autre part, nous constatons que les

propositions sont directement liées à des observations et une généralisation faites de la pratique.

Tableau CXXI

Lois empiriques candidates du génie logiciel

1. The law of continuing change
2. The law of increasing complexity
3. The more seen, the more needed
4. Change during development is inevitable
5. You can reuse without a big investment
6. Half the errors found in 15 percent of modules
7. A few good people are better than many less skilled people
8. People and time are not interchangeable
9. Software will continue to change
10. Software's entropy increases
11. The older a program, the more difficult it is to maintain
12. Language affect maintainability
13. Maintenance causes more errors than development

CHAPITRE 5

PHASE 3 – ANALYSE SELON LES CRITÈRES D'ENSEMBLE ET LIENS AVEC LA NORME ISO/IEC12207

5.1 Introduction

La phase 2 a permis de diminuer le nombre de propositions de 308 à 35. À celles-ci, quatre propositions ont été ajoutées, suite à une reformulation mineure, pour un total de 39 propositions. Le tableau CXXII présente les propositions retenues qui serviront d'entrée à la phase 3.

Tableau CXXII

Liste des propositions retenues de la phase 2

1. Align incentives for developer and customer	24. Maintain clear accountability for results
2. Apply and use quantitative measurements in decision making	25. Produce software in a stepwise fashion
3. Build software so that it needs a short user manual	26. Quality is the top priority; long term productivity is a natural consequence of high quality
4. Build with and for reuse	27. Rotate people through product assurance
5. Communicate with customers/users	28. Since change is inherent to software, plan for it and manage it
6. Define software artifacts rigorously	29. Since tradeoffs are inherent to software engineering, make them explicit and document it
7. Design for change	30. Strive to have a peer, rather than a customer, find a defect
8. Design for errors	31. Tailor cost estimation methods
9. Design for maintenance	32. To improve design, study previous solutions to similar problems
10. Determine requirements now	33. Use better and fewer people
11. Don't overstrain your hardware	34. Use documentation standards
12. Don't test your own software	35. Write programs for people first
13. Don't try to retrofit quality	
14. Don't write your own test plans	
15. Establish a software process that provides flexibility	
16. Fix requirements specification error now	
17. Give product to customers early	

Tableau CXXII (suite)

18. Give software tools to good engineers 19. Grow systems incrementally 20. Implement a disciplined approach and improve it continuously 21. Invest in the understanding of the problem 22. Involve the customer 23. Keep design under intellectual control	36. Know software engineering's techniques before using development tools 37. Select tests based on the likelihood that they will find faults 38. Choose a programming language according to maintainability 39. In face of unstructured code, rethink the module and redesign it from scratch.

5.2 Objectifs de la phase 3

Lors de la deuxième phase, nous avons appliqué les cinq critères individuels pour chacune des 308 propositions. La troisième phase a pour objectif premier de classer les propositions selon les catégories : processus, produit et individu. Le deuxième objectif est d'appliquer les deux critères d'ensemble aux 39 propositions retenues. Par la suite, les propositions de catégories processus seront liés aux processus de la norme ISO/IEC12207.

5.3 Méthode utilisée

Les 39 propositions issues de la phase 2 sont le principal intrant de la phase 3. À celles-ci s'ajoutent les deux critères d'ensemble. Les critères d'ensemble ne peuvent s'appliquer aux propositions prises individuellement. Ces critères doivent être appliqués sur l'ensemble des propositions. Afin d'obtenir une vision structurée de l'ensemble des propositions à analyser, les propositions seront catégorisées en fonction des trois catégories identifiées : processus, produit et individu. Ces catégories seront, au préalable, définies pour éviter les ambiguïtés. Une proposition peut être rattachée à plus d'une catégorie, le cas échéant. Ces regroupements faciliteront par la suite l'application des critères d'ensemble. Compte tenu que plusieurs propositions sont de catégorie

processus , l'utilisation de la classification des processus offerte par la norme ISO/IEC 12207 permettra de raffiner le classement des propositions de cette catégorie.

La figure 27 présente les principaux éléments de la méthode de recherche suivie pour la phase 3 de l'évaluation des propositions.

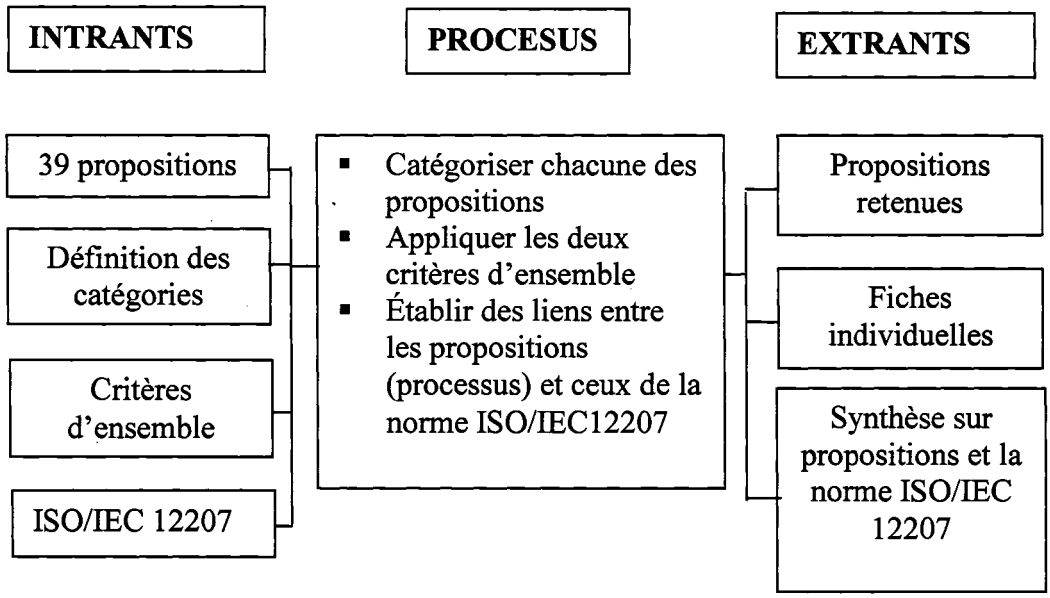

Figure 27 Méthode de recherche pour l'évaluation des propositions - phase 3

5.4 Définitions des catégories

Les catégories servant de base à l'analyse sont identifiées à la figure 28.

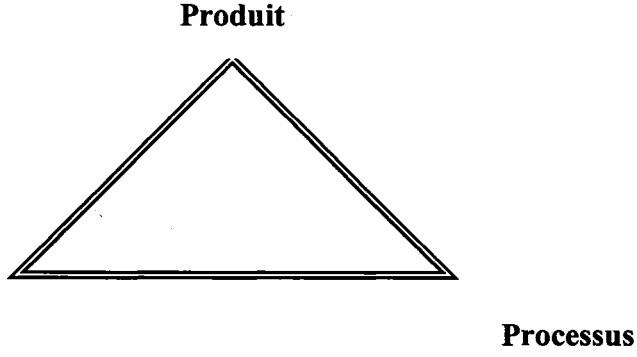

Figure 28 Catégories de principes

Ces catégories ont déjà été identifiées dans le chapitre sur les concepts à la base du génie. Chacune des catégories est maintenant définie.

5.4.1 Catégorie Produit

La norme ISO/IEC 12207 définit le terme produit (logiciel) comme suit : « *Set of computer programs, procedures, and possibly associated documentation and data* » (p.4).

Ghezzi et al. (2003) soulignent que le terme produit représente généralement ce qui est livré au client (« *end product* »), ce qui inclut le code exécutable et le guide de l'utilisateur. Cependant, ils précisent qu'il y a aussi des produits intermédiaires nommés « *work products* » conçus tout au long du processus de développement. Les produits intermédiaires comprennent, entre autres, le document des exigences logicielles, de conception, de tests, d'assurance qualité et de projet. Selon ces auteurs, les produits intermédiaires sont assujettis aux mêmes standards de qualité que les produits livrables au client. La gestion de configuration permet de maintenir une cohésion entre les différents produits intermédiaires et une version du produit final.

Nous considérons donc que le produit englobe tant le produit final (le logiciel) que les produits intermédiaires issus du processus du cycle de vie du logiciel. Ainsi, tout ce qui est sujet à la gestion de configuration sera considéré comme étant de catégorie produit.

5.4.2 Catégorie processus

Le Project Management Institute définit le terme processus comme suit : « *A series of actions bringing about a result* » (IEEE1490-2003). La norme ISO/IEC 12207 définit le terme processus comme étant : « *A set of interrelated activities which transform inputs into outputs* ». De plus, cette norme précise que les activités incluent l'utilisation de ressources pour transformer les intrants en extrants.

Comme les processus du génie logiciel sont nombreux et variés, il n'est pas suffisant, pour notre analyse, d'identifier qu'une proposition est de catégorie processus. Un niveau de granularité plus fin est requis. À cet effet, la structure de la norme ISO/IEC 12207 est intéressante. D'une part, elle offre un niveau de granularité permettant de préciser à quelle catégorie de processus se rattache la proposition. D'autre part, la norme est reconnue internationalement ; ainsi, on peut s'appuyer sur ses bases pour effectuer le raffinement des propositions de catégorie processus. Ce raffinement permettra de constater si plusieurs propositions couvrent, à titre d'exemple, les mêmes thèmes. Ainsi, il sera possible d'identifier si des propositions sont déduites ou si elles se contredisent. À la suite de la phase 3, il nous sera possible de constater le degré de couverture des propositions retenues selon la structure de la norme ISO/IEC 12207 et d'en tirer des conclusions.

La norme ISO/IEC 12207 présente les processus du cycle de vie utilisés pour acquérir un logiciel, le développer, le fournir à un client, l'exploiter et le maintenir. La norme identifie 17 processus principaux couvrant l'ensemble du cycle de vie du logiciel. Ces

processus sont regroupés sous trois catégories : processus primaires, de soutien et organisationnels.

Le tableau CXXIII présente les différentes catégories de processus du cycle de vie (ISO/IEC 12207).

Tableau CXXIII

Catégories de processus ISO/IEC 12207 (traduction Séguin)

Processus du cycle de vie ISO/IEC 12207		
Primaires	**De soutien**	**Organisationnels**
1. Acquisition 2. Approvisionnement 3. Développement 4. Exploitation 5. Maintenance	1. Documentation 2. Gestion des configurations 3. Assurance qualité 4. Vérification 5. Validation 6. Revue 7. Audit 8. Résolution des problèmes	1. Management 2. Infrastructure 3. Amélioration 4. Formation

Les processus primaires s'appliquent à l'entité (personne ou une organisation) qui procède au développement, à l'exploitation ou à la maintenance de produits logiciels. L'entité peut représenter un fournisseur, un acquéreur, un développeur, un opérateur ou un mainteneur (ISO/IEC12207). Le tableau CXXIV présente les cinq processus primaires.

Tableau CXXIV

Description des processus primaires ISO/IEC 12207 (adaptation Seguin)

Processus primaires	**Description et activités**	
1. Acquisition	Processus d'acquisition de produits logiciels (système, produit, service)	
	▪ Appel d'offre ▪ Préparation du contrat	▪ Suivi des fournisseurs ▪ Acceptation
2. Approvisionnement	Processus visant à fournir un produit ou un service logiciels	
	▪ Préparation d'appel d'offre ▪ Contrat ▪ Planification	▪ Exécution et suivi ▪ Revue et évaluation ▪ Livraison
3. Développement	Processus pour la définition et le développement de produits logiciels	
	▪ Analyse des exigences du système ▪ Architecture du système ▪ Analyse des exigences logicielles ▪ Conception architecturale du logiciel ▪ Conception détaillée ▪ Programmation et tests	▪ Intégration logicielle ▪ Test de qualification de logiciel ▪ Intégration système ▪ Test de qualification du système ▪ Installation du logiciel ▪ Acceptation du logiciel
4. Exploitation	Processus pour l'exploitation d'un système logiciel dans un environnement réel	
	▪ Test opérationnel ▪ Opération du système	▪ Soutien aux utilisateurs
5. Maintenance	Processus pour l'entretien du logiciel incluant aussi la migration et le retrait du logiciel	
	▪ Analyse des problèmes et des modifications ▪ Mise en place des modifications	▪ Revue et acceptation ▪ Migration ▪ Retrait du logiciel

Les processus de soutien viennent appuyer les processus primaires dans le but de favoriser le succès et la qualité du projet logiciel. Un processus de soutien est utilisé et exécuté par un autre processus. Le tableau CXXV présente les huit processus de soutien identifiés par la norme ISO/IEC12207.

Tableau CXXV

Description des processus de soutien ISO/IEC12207 (adaptation Seguin)

Processus de soutien	**Description et activités**	
1. Documentation	Processus régissant la documentation de l'information générée par les processus du cycle de vie	
	▪ Conception et développement	▪ Rédaction ▪ Mise à jour
2. Gestion de configuration	Processus de gestion de configuration des produits intermédiaires et finaux	
	▪ Identification des éléments ▪ Contrôle de la configuration	▪ Suivi de la configuration ▪ Évaluation de la configuration ▪ Gestion des versions et des livraisons
3. Assurance qualité	Processus qui assure que les produits et les processus sont conformes aux exigences et aux plans	
	▪ Conformité du produit ▪ Conformité du processus	▪ Conformité du système de qualité
4. Vérification	Processus déterminant si les produits issus d'une activité sont conformes aux exigences ou aux conditions imposées dans les activités antérieures	
	▪ Contrats ▪ Processus	▪ Exigences, Design ▪ Code, intégration et documentation
5. Validation	Processus permettant de valider si le produit final correspond aux attentes et aux exigences initiales	
	▪ Cas de tests ▪ Conformité des tests avec les exigences ▪ Effectuer les tests	▪ Valider la conformité du logiciel ▪ Tester le logiciel dans son environnement
6. Revue	Processus permettant d'évaluer l'état d'une activité et des produits qui en découlent	
	▪ Revues de gestion de projet	▪ Revues techniques

Tableau CXXV (suite)

Processus de soutien	**Description et activités**	
7. Audit	Processus permettant de déterminer la conformité des exigences, des plans et des contrats	
	▪ Revue de suivi de projet	▪ Revues techniques
8. Résolution de problème	Processus d'analyse et de résolution des problèmes découverts en cours de projet	
	▪ Description du problème ▪ Analyse du problème	▪ Solution au problème ▪ Mise en place

Les processus organisationnels permettent d'organiser et d'améliorer les autres processus. Les processus organisationnels sont au nombre de quatre, tels que présentés au tableau CXXVI.

Tableau CXXVI

Description des processus organisationnels ISO/IEC12207 (Adaption Séguin)

Processus organisationnels	**Description et activités**	
1. Management	Processus regroupant les activités de gestion du produit, du projet et des tâches	
	▪ Planification ▪ Déroulement et suivi	▪ Revue et évaluation ▪ Fin du projet
2. Infrastructure	Processus permettant d'établir et de maintenir l'infrastructure nécessaire au projet. L'infrastructure peut comprendre le matériel, le logiciel, les outils, les techniques, les normes, les aménagements pour le développement, l'opération et la maintenance.	
	▪ Mise en place de l'infrastructure	▪ Mise à jour de l'infrastructure
3. Amélioration	Processus qui permet d'établir, d'évaluer, de mesurer, de contrôler et d'améliorer les processus du cycle de vie	
	▪ Évaluation	▪ Amélioration

Tableau CXXVI (suite)

Processus organisationnels	**Description et activités**	
4. Formation	Processus qui permet de former le personnel et de maintenir leurs connaissances à jour	
	▪ Conception du matériel de formation	▪ Mise en place du plan de formation

5.4.3 Catégorie Individus

Wiegers (1996) affirme que les individus impliqués dans un projet logiciel ont un impact certain sur le succès du projet et sur la qualité du produit. Même si les processus sont clairement définis, ce sont les individus qui vont les interpréter et les mettre en œuvre.

Le PMBOK (2004) identifie des catégories d'intervenants à un projet. L'utilisation des catégories du PMBOK est pertinente puisque que SWEBOK se réfère au PMBOK pour tous les sujets concernant la gestion de projet qui ne sont pas traités au niveau de SWEBOK. La figure 29 présente les catégories d'intervenants tirées du PMBOK.

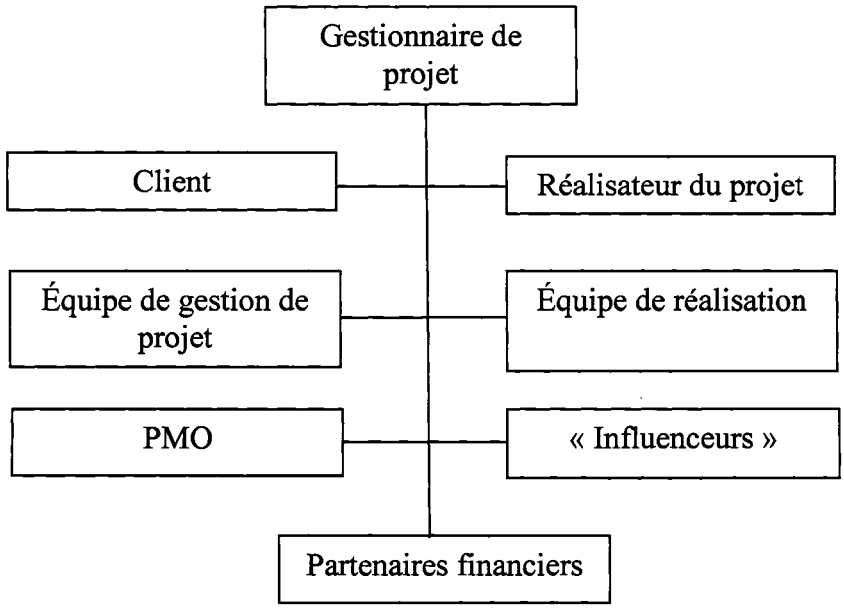

Figure 29 Catégories d'intervenants dans un projet PMBOK 2004 (adaptation Séguin)

Le gestionnaire de projet est l'individu responsable de la gestion du projet. Cette catégorie est explicitement identifiée puisque le responsable du projet n'est pas nécessairement un individu appartenant à la catégorie client ou à l'organisation qui réalise le projet. La catégorie « Client » représente les individus ou l'organisation qui utilisent le produit développé. Pour notre étude, les individus tels les utilisateurs directs du logiciel sont principalement ciblés. La catégorie « réalisateur » représente l'organisation dont les employés vont réaliser le projet. Comme il s'agit d'une entité organisationnelle, cette catégorie ne sera pas retenue, au profit de la catégorie « équipe de réalisation ». La catégorie « équipe de réalisation » comprend les individus qui feront le développement du produit. L'équipe de réalisation peut aussi être un groupe interne du client qui fera le développement du produit, tel un service de l'informatique. Cette catégorie comprend divers types d'individus, tels, entre autres, les analystes ou ingénieurs, les techniciens, les administrateurs de base de données. La catégorie « équipe de gestion de projet » représente les individus qui sont directement impliqués dans la gestion du projet. La catégorie « partenaires financiers » (Sponsor) représente les individus ou les organisations qui contribuent au soutien financier du projet. Pour notre étude, seul le volet individu sera considéré. La catégorie des « influenceurs » représente les individus ou les groupes appartenant soit au groupe client ou au groupe réalisateur qui ne sont pas directement liés au produit, mais qui peuvent influencer positivement ou négativement le déroulement du projet. La catégorie Project Management Office (PMO) représente une entité organisationnelle responsable de la gestion des différents projets. Le PMBOK souligne que le PMO peut être un intervenant direct dans le projet s'il a une influence sur le résultat du projet. Comme le PMO est une entité organisationnelle et non un individu, cette catégorie ne sera pas retenue.

Les catégories retenues du PMBOK serviront à identifier le rattachement d'un principe de catégorie individu.

5.5 Critères d'ensemble

Au chapitre 3, nous avons établi sept critères d'identification des principes: cinq critères d'identification individuels et deux critères d'identification d'ensemble. La phase 2 a appliqué les critères individuels sur les 308 propositions. Les deux critères d'ensemble seront maintenant appliqués sur l'ensemble des 39 propositions retenues suite à la phase 2. Les deux critères d'ensemble sont les suivants :

1. Les principes devraient être indépendants (non déduits) (Boehm 1983)
La définition du terme principe souligne qu'un principe est une proposition non-déduite. Ainsi, ce critère permettra de mettre en force ce volet de la définition retenue pour l'étude. Si une proposition est déduite d'une autre proposition, alors la proposition ne peut pas être considérée comme fondamentale.

2. Un principe ne doit pas contredire un autre principe connu. (Bourque et al. 2002)
Ce critère permettra d'écarter des propositions qui seraient en contradiction. En cas de contradiction entre deux propositions, celles-ci seront isolées et un arbitrage devra être fait.

Pour chacune des trois catégories (individus, produit et processus), un tableau sera produit présentant la concordance ou non de chacune des propositions selon les deux critères. Le tableau CXXVII présente le format type qui sera utilisé.

Tableau CXXVII

Format type de présentation de concordance entre les propositions et les critères

No.	Proposition	Critère 1	Critère 2
1	Align incentives for developer and customer	√	√
5	Communicate with customers/users		√
18	Give software tools to good engineers		√

Le premier élément est le numéro de la proposition. Un numéro a été assigné à chacune des 39 propositions retenues à la phase 2. Ce numéro fait référence au tableau 1 présenté au début de ce chapitre. La proposition est par la suite retranscrite pour faciliter la lecture du tableau. Il est à noter que si la proposition originale d'un auteur a été reformulée, c'est la reformulation qui sera utilisée à la phase 3. Les deux colonnes suivantes indiquent si les critères sont satisfaits ou non. Le critère no. 1 représente l'indépendance du principe et le critère no. 2 représente le fait que le principe n'en contredit pas un autre. Le « √ » indique que la proposition satisfait le critère. La case de couleur rouge indique que le critère n'est pas satisfait. Dans le cas du critère no.1, en plus de la couleur rouge, le numéro de la proposition « parent » est indiqué. Ainsi, la proposition évaluée est alors déduite de la proposition « parent ».

5.6 Résultat de la phase 3

Dans un premier temps les 39 propositions ont été classées selon les catégories: produit, individu et processus. Ainsi, la catégorie individus recueille six propositions, la catégorie produit sept et 36 propositions pour la catégorie processus, tel que présenté au tableau CXXVIII.

Tableau CXXVIII

Ventilation des propositions selon les catégories

Catégories	**Nombre de propositions**
Individu	6
Produit	7
Processus	36

Le nombre total de propositions est supérieur à 39 compte tenu que certaines se retrouvent dans plus d'une catégorie. Dans les prochaines sections, nous présenterons pour chacune des catégories les résultats de l'analyse.

5.6.1 Catégorie individu

La catégorie individus regroupe six propositions tel que présenté au tableau CXXIX.

Tableau CXXIX

Liste des propositions rattachées à la catégorie Individu

No.	Proposition	Critère 1	Critère 2
1	Align incentives for developer and customer	√	√
5	Communicate with customers/users		√
18	Give software tools to good engineers		√
26	Quality is the top priority; long term productivity is a natural consequence of high quality	√	√
33	Use better and fewer people	√	√
36	Know software engineering techniques before using development tools	√	√

Proposition 1: *Align incentives for developer and customer*

La proposition no. 1, selon le sens donné par Davis 1995, suggère que les développeurs soient gratifiés ou pénalisés selon l'atteinte ou non des objectifs. L'explication donnée par l'auteur vise essentiellement les développeurs, malgré la présence du terme client dans la proposition. Cette proposition implique les individus du groupe "équipe de réalisation". La proposition satisfait les deux critères, elle est donc retenue.

Proposition 5: *Communicate with customers/users*

La proposition no. 5 suggère d'assurer une bonne communication avec le client. Cette proposition peut être déduite de la proposition no. 22 ("Involve the customer") qui est plus générale. Cette dernière étant plus générale peut être considérée comme plus fondamentale que la proposition no. 5. La proposition no. 5 implique les individus des groupes client, gestionnaire de projet et équipe de réalisation. La proposition no. 5 n'ajoute pas d'éléments nouveaux ou de précisions par rapport à la proposition père. La proposition n'est donc pas retenue.

Proposition 18: *Give software tools to good engineers*

La proposition no. 18 souligne que les outils de développement (ex : CASE) devraient êtres fournies aux développeurs compétents. Davis (1995) souligne qu'un individu ne maitrisant pas les techniques de base du génie logiciel ne peut tirer profit des avantages de ces outils, ni même les utiliser adéquatement. Cette proposition peut être déduite de la proposition plus générale no. 36 ("Technique before tools"). Cette dernière a été reformulée comme suit à la phase 1 : "Know software engineering techniques before using development tools". La proposition no. 18 n'ajoute pas de précision supplémentaire par rapport à la proposition no.36. La proposition no. 18 n'est donc pas retenue.

Proposition 26: *Quality is the top priority; long term productivity is a natural consequence of high quality*

Wiegers (1996) souligne que l'attitude des développeurs et des gestionnaires envers la qualité a un impact certain sur la qualité des processus et du produit final. L'attitude des individus envers la qualité est un élément déterminant que même les processus de qualité ne peuvent substituer. La proposition n'est pas déduite, ni en contradiction avec d'autres. Elle est donc retenue.

Proposition 33: *Use better and fewer people*

La proposition no. 33 se retrouve dans deux catégories : individu et processus. Au niveau de la catégorie individu, la proposition est un guide pour le gestionnaire du projet lorsqu'il fait l'allocation des ressources humaines d'un projet. La proposition satisfait les deux critères et elle est donc retenue.

Proposition 36: *Know software engineering techniques before using development tools*

La proposition no. 36 vise principalement les individus de l'équipe de réalisation qui doivent maitriser les techniques de base du génie logiciel avant d'utiliser les outils de développement. Ces individus pourront ainsi mieux tirer profit des possibilités des outils de développement. La proposition satisfait les deux critères et elle est donc retenue.

5.6.1.1 Sommaire de la catégorie individu

Nous constatons qu'aucune des propositions de la catégorie individu n'est en contradiction avec une autre proposition. Les propositions no.5 et no.18 sont déduites de propositions plus générales. Des six propositions de la catégorie individu, quatre sont retenues. Le tableau CXXX présente celles-ci.

Tableau CXXX

Propositions retenues de la catégorie Individu

No.	**Propositions**
1.	Align incentives for developer and customer
26.	Quality is the top priority; long term productivity is a natural consequence of high quality
33.	Use better and fewer people
36.	Know software engineering's techniques before using development tools

Le tableau CXXXI présente une ventilation des propositions de la catégorie individu en fonction des groupes du PMBOK.

Tableau CXXXI

Ventilation des propositions selon les groupes d'intervenants du PMBOK

Groupes d'intervenants	**Propositions associées**
Client	Aucune
Gestionnaire de projet	#26 et #33
Équipe de gestion de projet	Aucune
Équipe de réalisation	#1, #26, et #36
PMO	Aucune
Influenceurs	Aucune
Partenaires	Aucune

Nous constatons que seulement deux groupes sur les sept retenus sont associés à des propositions de la catégorie individus. Les groupes client, équipe de projet, PMO, influenceurs et partenaires sont vacants. Seuls les groupes gestionnaire de projet et équipe de réalisation sont associés à des propositions. Nous constatons que les propositions visent en premier les développeurs et par la suite le gestionnaire du projet. Tous les autres groupes d'intervenants ne sont pas associés aux propositions retenues de la catégorie individu. Nous soulignons que le groupe client reste vacant, groupe ayant pourtant une importance certaine dans un projet logiciel.

5.6.2 Catégorie produit

La catégorie produit regroupe sept propositions, tel que présenté au tableau CXXXII. Ces propositions sont associées aux produits intermédiaires du processus de développement ou au produit final.

Tableau CXXXII

Résultat de l'application des critères aux propositions de la catégorie produit

No.	Propositions	Critère 1	Critère 2	Type produit
3.	Build software so that it needs a short user manual	√	√	Final
4.	Build with and for reuse	√	√	Final
6.	Define software artifacts rigorously	√	√	Intermédiaire
11.	Don't overstrain your hardware	√	√	Final
34.	Use documentation standards		√	Intermédiaire
35.	Write programs for people first	√	√	Final
38.	Choose a programming language to assure maintainability	√	√	Final

Proposition 3: *Build software so that it needs a short user manual*

La proposition no. 3 concerne le produit final et plus précisément la facilité d'utilisation du logiciel. Davis (1995) affirme que plus le manuel d'utilisation est petit, meilleur est le logiciel. Nous considérons que cette affirmation est nettement exagérée puisqu'il y a de nombreuses caractéristiques de qualité du logiciel (ISO9126) qui ne sont pas visibles de l'extérieur, ni à l'utilisation. Néanmoins, la proposition satisfait les deux critères et elle est retenue.

Proposition 4: *Build with and for reuse*

La proposition no. 4 (Bourque et al. 2002) n'est pas commentée par les auteurs. Nous interprétons donc cette proposition à l'effet de construire le logiciel en réutilisant des composants existants. Ces composants doivent être conçus pour favoriser la réutilisation. La proposition est directement liée au logiciel, donc au produit final. Cette proposition satisfait les deux critères et elle est donc retenue.

Proposition 6: *Define software artifacts rigorously*

La proposition no. 6 (Bourque et al. 2002) n'est également pas commentée par les auteurs. Le terme central de la proposition est « artifact ». Nous considérons que les artéfacts représentent tous les produits intermédiaires générés par le processus de développement. Les produits intermédiaires doivent être définis rigoureusement. Cette proposition satisfait les deux critères et elle est donc retenue.

Proposition 11: *Don't overstrain your hardware*

La proposition no. 11 souligne que si le matériel est surchargé par le logiciel, il y aura beaucoup d'efforts de consentis pour ajuster le logiciel aux contraintes du matériel. Ces efforts auront un impact sur les coûts et les délais de développement. La proposition concerne le produit final, le logiciel. Cette proposition satisfait les deux critères et elle est donc retenue.

Proposition 34: *Use documentation standards*

La proposition no. 34 souligne l'utilisation de normes de documentation. Ces normes peuvent s'appliquer au produit final (manuel d'utilisation) et aux différents produits intermédiaires. La proposition est déduite de la proposition no. 20 plus générale qui stipule qu'il faut implémenter un processus discipliné et de l'améliorer constamment. L'utilisation de normes est des éléments d'une approche disciplinée. Malgré que la proposition puisse être déduite de la proposition no. 20, elle ajoute une précision que n'offre pas la proposition parent, à l'effet d'utiliser concrètement des normes de documentation. Également, l'application de la proposition permet de mieux soutenir la proposition no. 6, à l'effet de définir rigoureusement les artéfacts du logiciel. Cependant, on ne peut conclure que la proposition no. 6 est déduite de la no. 34, car celle-ci se

concentre spécifiquement sur les normes de documentation. D'autres normes peuvent être utilisées pour soutenir la proposition no. 6. La proposition no. 34 est donc retenue.

Proposition 35: *Write programs for people first*

La proposition no. 35 souligne l'importance de la lisibilité du code afin que d'autres personnes puissent, par la suite, facilement le lire et le modifier. Les programmes sont des produits et la lisibilité de ceux-ci est une caractéristique du produit final. Cette proposition satisfait les deux critères et elle est donc retenue.

Proposition 38: *Choose a programming language according to maintainability*

La proposition no. 38, « Language affect maintability » a été reformulée à la phase 2 de l'analyse comme suit : « Choose a programming language to assure maintainability». Selon Davis (1995), les langages ne sont pas tous égaux au niveau de la facilité de maintenance des programmes. Ainsi, lors du choix d'un langage de programmation, les considérations de maintenance devraient être prises en compte. Cette proposition se classe sous deux catégories. Le choix comme tel du langage est une étape dans le processus de développement tandis que la facilité de maintenance d'un logiciel est une des caractéristiques du produit final. Cette proposition satisfait les deux critères et elle est donc retenue.

5.6.2.1 Sommaire de la catégorie produit

Suite à l'application des critères d'ensemble sur les propositions de la catégorie produit, toutes les propositions sont retenues tel que présenté au tableau CXXXIII.

Tableau CXXXIII

Propositions retenues de la catégorie produit

No.	Propositions
3	Build software so that it needs a short user manual
4	Build with and for reuse
6	Define software artifacts rigorously
11	Don't overstrain your hardware
34	Use documentation standards
35	Write programs for people first
38	Choose a programming language to assure maintainability

Une seule proposition (no. 34) peut être déduite d'une autre. Cependant, la formulation de la proposition no. 34 ajoute une précision permettant de mieux l'appliquer en guidant une action précise. La proposition no. 34 a donc été conservée, malgré une déduction possible.

Le tableau CXXXIV présente la répartition des propositions en fonction du type de produit.

Tableau CXXXIV

Ventilation des propositions de catégorie produit

Type de produit	Propositions
Intermédiaire	No.6 et 34
Final	No.3, 4, 11, 35 et 38

Nous constatons que les propositions retenues visent principalement le produit final (cinq sur sept) et que seulement deux propositions sont associées aux produits intermédiaires. Nous soulignons qu'il y a possiblement une carence de propositions concernant les produits intermédiaires qui ont une grande importance dans le déroulement du processus de développement.

5.6.3 Catégorie processus

La catégorie processus regroupe la majorité des propositions de cette étude, soit 36 sur 39 propositions. Le tableau CXXXV présente la liste des propositions classées dans cette catégorie.

Tableau CXXXV

Résultat de l'application des critères aux propositions de la catégorie processus

No.	Proposition	Critère 1	Critère 2
1	Align incentives for developer and customer	√	√
2	Apply and use quantitative measurements in decision making	√	√
3	Build software so that it needs a short user manual	√	√
4	Build with and for reuse	√	√
5	Communicate with customers/users		√
6	Define software artifacts rigorously	√	√
7	Design for change		√
8	Design for errors	√	√
9	Design for maintenance		√
10	Determine requirements now		√
12	Don't test your own software		√
13	Don't try to retrofit quality	√	√
14	Don't write your own test plans		√
15	Establish a software process that provides flexibility	√	√
16	Fix requirements specification error now		√
17	Give product to customers early		√
18	Give software tools to good engineers		√
19	Grow systems incrementally	√	√
20	Implement a disciplined approach and improve it continuously	√	√
21	Invest in the understanding of the problem	√	√
22	Involve the customer	√	√
23	Keep design under intellectual control		√
24	Maintain clear accountability for results		√
25	Produce software in a stepwise fashion		√

Tableau CXXXV (suite)

No.	Proposition	Critère 1	Critère 2
26	Quality is the top priority; long term productivity is a natural consequence of high quality	√	√
27	Rotate people through product assurance	√	√
28	Since change is inherent to software, plan for it and manage it	√	√
29	Since tradeoffs are inherent to software engineering, make them explicit and document it		√
30	Strive to have a peer, rather than a customer, find a defect	√	√
31	Tailor cost estimation methods	√	√
32	To improve design, study previous solutions to similar problems	√	√
33	Use better and fewer people	√	√
35	Write programs for people first	√	√
37	Select tests based on the likelihood that they will find faults	√	√
38	Choose a programming language according to maintainability	√	√
39	In face of unstructured code, rethink the module and redesign it from scratch.		√

En plus de vérifier si les propositions satisfont aux deux critères d'ensemble, des liens seront aussi faits avec les processus identifiés par la norme ISO/IEC 12207.

Nous observons que sur les 36 propositions, 14 propositions sont déduites d'autres propositions plus générales. Ces 14 propositions ne satisfont pas le critère no. 1. Par contre, toutes les propositions satisfont le critère no. 2. Comme nous le verrons, certaines propositions ont été conservées malgré qu'elles soient déduites d'autres propositions puisqu'elles ajoutent ou précisent d'avantage un élément par rapport à la proposition parent.

Nous allons commenter pour chacune des 36 propositions le résultat de notre analyse.

Proposition no.1: *Align incentives for developer and customer*

Nous avons déjà discuté de cette proposition qui se retrouve également dans la catégorie individu. Davis (1995) souligne que les exigences doivent être priorisées avec le client afin que les développeurs puissent par la suite percevoir leur importance relative. Nous interprétons l'explication donnée par Davis à l'effet qu'il cible essentiellement le développement du code et non les étapes précédentes ou subséquentes à la construction du logiciel. La proposition satisfait aux deux critères, elle est donc retenue.

Cette proposition est liée aux processus primaires de développement, mais il serait possible d'étendre sa portée aux activités précédant le code jusqu'aux activités d'installation du logiciel. Cependant, Davis (1995) a strictement limité la portée de ce principe à la phase de programmation.

La mise en place des incitatifs et de leur vérification relèvent des processus de type organisationnel, particulièrement du management. La gestion de projet est responsable de la mise en place des incitatifs et du suivi de ceux-ci. Le tableau CXXXVI présente les relations entre la proposition et les processus de la norme ISO/IEC12207.

Tableau CXXXVI

Proposition no.1 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	 5.3.7 Software coding and testing
Organisationnel 7.1 Management	 7.1.2 Planning 7.1.3 Execution and control 7.1.4 Review and evaluation

Proposition no.2: *Apply and use quantitative measurements in decision making*

Bourque et al. (2002) ne commentent pas cette proposition. Une interprétation possible est à l'effet que le contrôle (le management) devrait procéder à la prise de mesure au

niveau des processus et d'utiliser les résultats obtenus pour la prise de décision. Le contrôle peut difficilement être efficace sans l'utilisation de mesures. Cette proposition est principalement associée au processus organisationnel de management. Cependant, nous soulignons que cette proposition peut aussi englober des processus de développement et que certaines décisions, nécessitant des données quantitatives, peuvent être prises par les développeurs.

Tableau CXXXVII

Proposition no.2 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Organisationnel 7.1 Management	 7.1.3 Execution and control 7.1.4 Review and evaluation

Proposition no.3: *Build software so that it needs a short user manual*

La proposition no. 3 a été analysée au niveau de la section des propositions de catégorie produit. Au niveau de la catégorie processus, c'est le terme « Build » implique explicitement le processus de développement du logiciel. Le processus doit tenir compte de l'objectif que le logiciel développé devrait être facile à utiliser sans nécessité un volumineux guide d'utilisation. L'explication donnée par l'auteur vise exclusivement la construction du logiciel, malgré le fait que d'un point de vue général, elle pourrait aussi toucher à la maintenance et à l'exploitation du logiciel. La proposition satisfait les deux critères et elle est retenue. La proposition est liée aux processus primaires de développement, tel que présenté au tableau CXXXVIII.

Tableau CXXXVIII

Proposition no.3 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.7 Software coding and testing

Proposition no.4: *Build with and for reuse*

Cette proposition est aussi membre de la catégorie produit. En effet, c'est une qualité de certains composants logiciels d'être réutilisables. Cet aspect est couvert par la partie "for reuse" de la proposition. Au niveau de la catégorie processus, c'est la portion "Build with" qui est ciblée. Construire un logiciel en réutilisant des composants déjà faits est un aspect du processus qui doit être pris en compte au niveau de la conception et de la construction du logiciel. La proposition no. 4 satisfait les deux critères, elle est donc retenue.

La proposition est liée aux processus primaires de développement, tel que présenté au tableau CXXXIX.

Tableau CXXXIX

Proposition no.4 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.7 Software coding and testing

Proposition no.5: *Communicate with customers/users*

Cette proposition peut être déduite de la proposition plus générale no. 22 ("Involve the customer"). La proposition no. 5 n'apporte gère plus de précision à la proposition père no. 22. Si le client s'implique dans le projet, la communication avec le client est implicite. De ce fait, la proposition no. 5 n'est donc pas retenue.

Proposition no.6: *Define software artifacts rigorously*

Cette proposition a été commentée au niveau de la catégorie produit. Concernant la catégorie processus, c'est le terme « *define* » qui a une implication directe sur le processus. De plus, le terme « *rigorously* » ajoute une contrainte sur le processus. La proposition satisfait les deux critères, elle est donc retenue.

La proposition est liée aux processus primaires de développement, tel que présenté au tableau CXL.

Tableau CXL

Proposition no.6 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.7 Software coding and testing

Proposition no.7: *Design for change*

Davis (1995) souligne que la facilité de modifier le logiciel pour l'adapter au changement dépend en bonne partie des choix faits au niveau du design. L'auteur cite

des caractéristiques de qualité du design qui favoriseraient la facilité d'adapter par la suite le logiciel aux changements. Entre autres, le design doit être modulaire afin qu'un changement apporté à un module ait un impact mineur sur les autres modules du logiciel. Le design devrait être portable afin qu'il soit adaptable à des changements de matériel et de système d'exploitation. Le design doit être aussi malléable afin de permettre l'ajout de nouvelles exigences.

La proposition no. 7 est déduite de la proposition no. 9 (Design for maintenance). La portée de la proposition no. 9 englobe la proposition no. 7 et en plus tous les correctifs faits au logiciel tel les défauts et l'amélioration des performances. Comme le processus de maintenance inclut les changements apportés au logiciel, ainsi que les correctifs, nous ne considérons pas que la proposition no. 7 ajoute d'élément nouveau à la proposition no. 9. La proposition no. 7 n'est donc pas retenue.

Proposition no.8: *Design for errors*

Davis (1995) souligne qu'à l'étape du design, il serait requis de s'assurer que des erreurs ne soient pas introduites dans le design. Le cas échéant, celles-ci devraient être détectées et corrigées. Davis propose des actions à prendre qui sont reliés plutôt à la construction du logiciel et à son code. Nous considérons, d'une part que l'auteur utilise le terme design dans la formulation de la proposition, mais que les actions à prendre sont plutôt de nature de codage du logiciel. D'autre part, la formulation même de la proposition est boiteuse, car dans les faits on ne fait pas de design pour des erreurs. Ainsi, l'action suggérée est différente de la formulation de la proposition. Pour ces raisons, la proposition no. 8 sera écartée.

Proposition no.9 : *Design for maintenance*

Cette proposition ressemble à la proposition no. 7 qui est déduite de celle-ci. Adapter le logiciel au changement est un des éléments couverts par le processus de maintenance. La proposition no. 9 est déduite à son tour (figure 30) de la proposition no. 28 (Since change is inherent to software, plan for it and manage it). Cependant, la proposition no. 9 précise que le design doit prévoir les activités de la maintenance (évolutive et corrective). La proposition est donc retenue.

Figure 30 Proposition déduite de la proposition no.9

La proposition no. 9 est liée au processus du type primaire de la norme ISO/IEC12207, en particulier aux processus de développement et de maintenance tel que présenté au tableau CXLI.

Tableau CXLI

Proposition no.9: relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Développement	5.3.5 Software architectural design 5.3.6 Software detailed design
5.5 Maintenance	5.5.2 Problem and modification analysis 5.5.3 Modification implementation 5.5.4 Maintenance review/acceptance

Proposition no.10: *Determine requirements now*

Davis (1995) souligne que les exigences devraient être définies avant d'entreprendre les étapes subséquentes. L'auteur mentionne également des techniques pour élucider les exigences telles le prototypage. La proposition no. 10 peut se déduire de la proposition no. 21 qui est plus générale (Invest in the understanding of the problem). Malgré cette déduction possible, nous considérons que la proposition no. 10 précise l'action à faire afin de mieux comprendre le problème à résoudre. La définition des exigences logicielles représente un moyen de comprendre le problème. La proposition no. 21 ne précise pas les moyens ou l'action à prendre pour comprendre le problème. La proposition no. 10 est donc retenue.

Au niveau des processus de la norme ISO/IEC12207, la proposition no. 10 est liée aux processus de type primaire de développement tel que présenté au tableau CXLII.

Tableau CXLII

Proposition no.10 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.2 System requirement analysis 5.3.4 Software requirement analysis

Proposition no.12: *Don't test your own software*

Par cette proposition, Davis (1995) mentionne que les développeurs ne devraient pas tester le logiciel qu'ils ont développé, sauf pour les tests unitaires. Cette proposition peut être déduite de la proposition no. 30 (*Strive to have a peer, rather than a customer, find a defect*). La proposition no. 30 souligne qu'il est préférable que les pairs détectent les défauts plutôt que le client. Il est donc sous-entendu que les pairs effectuent les tests. C'est précisément cet aspect qui est en lien avec la proposition no. 12. La proposition

no. 12 n'ajoute pas d'éléments nouveaux par rapport à la proposition no. 30. Elle n'est donc pas retenue.

Proposition no.13 : *Don't try to retrofit quality*

Davis (1995) souligne que la qualité ne peut s'ajouter au logiciel en fin du processus. La qualité s'introduit à chacune des étapes du processus de développement par des activités spécifiques. La proposition satisfait aux deux critères. Elle est donc retenue.

La proposition est liée aux processus de soutien de la norme ISO/IEC 12207, particulièrement aux processus d'assurance qualité tel que présenté au tableau CXLIII.

Tableau CXLIII

Proposition no.13 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Soutien	
6.3 Quality assurance	6.3.2 Product assurance 6.3.3 Process assurance
6.4 Verification	6.4.2 Verification 6.4.2.2 Process verification 6.4.2.3 Requirements verification 6.4.2.4 Design verification 6.4.2.5 Code verification 6.4.2.6 Integration verification 6.4.2.7 Documentation verification
6.5 Validation	6.5.2 Validation

Proposition no.14: *Don't write your own test plans*

Davis (1995) souligne que les développeurs ne devraient pas concevoir les plans de tests pour le logiciel qu'ils ont développé. L'auteur affirme que les développeurs pourraient faire les même erreurs ou omissions qu'ils ont introduites au niveau du logiciel. La proposition no. 14 s'apparente à la proposition no. 12 que nous avons écartée antérieurement. La proposition no. 12 stipulait de confier les tests à des pairs, comme la proposition no. 30 le souligne. Par contre au niveau de la proposition no. 14, il est souligné que les développeurs ne doivent pas faire non plus la conception des plans de tests de leur logiciel. Nous considérons que la proposition no. 14 ajoute une précision intéressante (plan de tests) par rapport à la proposition no. 30. Elle est donc retenue

La proposition no. 14 est liée aux processus de type primaire de développement et de maintenance de la norme ISO/IEC12207, tel que présenté au tableau CXLIV.

Tableau CXLIV

Proposition no.14 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Développement	5.3.7 Software coding and testing 5.3.9 Software qualification testing 5.3.11 System qualification testing
5.5 Maintenance	5.5.3 Modification implementation 5.5.3.2 a)

Proposition no.15: *Establish a software process that provides flexibility*

Bourque et al. (2002) ne fournissent pas d'explication pour cette proposition. Nous l'interprétons donc comme suit : le processus mis en place doit pouvoir s'adapter (flexibilité) au contexte spécifique des projets et de l'organisation.

Nous devons vérifier si la flexibilité mentionnée dans la proposition entre en contradiction avec l'approche disciplinée de la proposition no.20. Nous considérons qu'un processus peut être discipliné tout en étant adaptable au contexte spécifique de l'entreprise. Il n'y a donc pas de contradiction entre la flexibilité et la discipline. La proposition satisfait aux deux critères et elle est donc retenue.

Au niveau de la norme ISO/IEC 12207, la proposition ne s'adresse pas à un processus en particulier identifié par la norme. Cependant, la norme souligne en introduction: « *The international standard is[...] designed to be tailored for an individual organization, project, or application. » (p.x).* La proposition a donc une portée globale sur la norme, mais pas pour un processus en particulier.

Proposition no.16 : *Fix requirements specification error now*

Davis (1995) souligne que les erreurs au niveau des exigences pour le logiciel doivent être détectées et corrigées dès cette étape. Une erreur provenant des exigences qui serait découverte plus tard dans le processus peut entraîner des coûts de correction importants.

Cette proposition peut être déduite de la proposition de la proposition no. 10 (Determine requirement now). La déduction provient du fait que l'étape de définition des exigences devrait inclure des activités de vérification et de validation. Cependant, la proposition no. 16 mentionne explicitement de corriger les erreurs dans les exigences. Même si elle est déduite, la proposition no. 16 sera retenue car elle ajoute une précision à la proposition no. 10.

La proposition no. 16 est liée aux processus de type primaire de développement et aux processus de soutien tel que présenté au tableau CXLV.

Tableau CXLV

Proposition no.16 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Développement	5.3.2 System requirement analysis 5.3.4 Software requirement analysis
Soutien	
6.4 Verification	6.4.2.3 Requirement verification

Proposition no.17 : *Give product to customers early*

Davis (1995) précise que le client peut s'impliquer dans le processus en utilisant des versions prototypes du logiciel. Ces prototypes permettent, entre autres, de valider les besoins exprimés par le client et d'obtenir ses commentaires. Par la suite, les exigences du logiciel seront complétées et le produit final pourra être développé.

Cette proposition est déduite de la proposition no. 22 (Involve the customer) qui plus générale (figure 31). Par contre, la proposition no. 17 précise de quelle façon le client peut s'impliquer dans le projet. La proposition est donc retenue car elle ajoute une précision par rapport à la proposition père.

22 : Involve the customer
17 : Give product to customers early

Figure 31 Proposition déduite de la proposition no.22

La proposition est liée aux processus primaires et de soutien de la norme ISO/IEC12207 tel que présenté au tableau CXLVI.

Tableau CXLVI

Proposition no.17 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement Soutien 6.5 Validation	5.3.4 Software requirement analysis

Proposition no.18: *Give software tools to good engineers*

Davis (1995) souligne que l'utilisation d'un outil de développement ne transforme pas un développeur médiocre en un excellent développeur. Tout comme l'utilisation d'un traitement de texte ne fait pas d'une personne un auteur de renom. L'auteur souligne que les outils (CASE) permettent à un développeur qui connaît bien les techniques du génie logiciel d'être plus performant. Par contre, les mêmes outils donnés à des développeurs ayant des carences au niveau de leurs connaissances ne rendront pas ceux-ci plus performants.

La proposition peut se déduire (figure 32) de la proposition no. 36 (Know software engineering's techniques before using development tools).

36 : Know software engineering's techniques before using development tools
18 : Give software tools to good engineers

Figure 32 Proposition déduite de la proposition no.36

Cette dernière souligne que le développeur devrait connaître les techniques du génie logiciel avant d'utiliser des outils de développement tel les outils CASE. Tout comme mentionné antérieurement au niveau de la catégorie individu, la proposition no. 18 n'ajoute pas d'éléments nouveaux par rapport à la proposition no. 36. La proposition no. 18 n'est donc pas retenue.

Proposition no.19: *Grow systems incrementally*

Davis (1995) souligne que le développement d'un système devrait se faire par l'ajout successif de fonction, soit de façon incrémentale. Davis favorise la mise en place d'un système d'envergure réduite, mais fonctionnel plutôt qu'attendre que toutes les fonctionnalités soient développées et procéder à une implantation globale de l'ensemble du système. Cette approche permet aussi de réduire le risque du projet. La proposition no. 19 n'est pas déduite et pas en contradiction avec une autre proposition. Elle est donc retenue.

Au niveau des processus de la norme ISO/IEC12207, la proposition est liée aux processus primaires de développement et particulièrement à effectuer des micro-cycles des activités 5.3.4 à 5.3.12 tel que présenté au tableau CXLVII.

Tableau CXLVII

Proposition no.19 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.8 Software integration 5.3.9 Software qualification testing 5.3.10 System integration 5.3.11 System qualification testing 5.3.12 Software installation

Proposition no.20: *Implement a disciplined approach and improve it continuously*

Bourque et al. (2002) ne fournissent pas d'explication sur cette proposition. Nous interprétons donc celle-ci à l'effet de mettre en place un processus discipliné de développement et de procéder à son amélioration d'une façon continue. L'utilisation du terme discipliné souligne une forme de rigueur dans le déroulement et dans la séquence des activités de développement. La proposition no. 20 est de nature générale et cinq autres propositions peuvent en être déduites tel que présenté à la figure 33. La proposition satisfait aux deux critères et elle est donc retenue.

20 : Implement a disciplined approach and improve it continuously
6 : Define software artifacts rigorously
23 : Keep design under intellectual control
24 : Maintain clear accountability for results
25 : Produce software in a stepwise fashion
34 : Use documentation standards

Figure 33 Propositions déduites de la proposition no.20

Toutes les propositions déduites comportent une forme de rigueur, donc de discipline à suivre.

La proposition no. 20 n'est pas liée à un processus en particulier de la norme ISO/IEC12207. La portée de cette proposition est plutôt de niveau méta. Tout comme la proposition no. 15, la proposition no. 20 est de niveau général. Par contre, l'utilisation de la norme permet de mettre en pratique la proposition no. 20.

Proposition no.21: *Invest in the understanding of the problem*

Bourque et al. (2002) ne décrivent pas le sens de cette proposition. Nous l'interprétons comme suit : avant de développer une solution logicielle, il est préférable de bien comprendre le problème et de prendre les actions nécessaires pour l'élucider. La

proposition n'est pas déduite, mais d'autres propositions sont déduites de celle-ci tel que présenté à la figure 34.

21 : Invest in the understanding of the problem
    10 : Determine requirements now
        16 : Fix requirements specification error now

Figure 34 Propositions déduites de la proposition no.21

La proposition 21 n'est pas déduite et ni en contradiction avec d'autres. Elle est donc retenue.

La proposition est liée à plusieurs processus de la norme ISO/IEC 12207, tel que présenté au tableau CXLVIII. La proposition n'implique pas seulement les processus reliés au développement, mais aussi ceux de l'acquisition de produits et de services, ainsi qu'à l'approvisionnement.

Tableau CXLVIII

Proposition no.21 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.1 Acquistion	5.1.1 Initiation
5.2 Approvisionnement	5.2.1 Initiation 5.2.2 Preparation of response
5.3 Développement	5.3.2 System requirement analysis 5.3.3 System architectural design 5.3.4 Software requirement analysis
5.5 Maintenance	5.5.2 Problem and modification analysis

Tableau CXLVIII (suite)

Processus	**Activités**
Soutien 6.8 Problem resolution	6.8.2 Problem resolution
Organisationnel 7.1 Management	7.1.3 Execution and control

Proposition no.22: *Involve the customer*

Royce (1970) souligne que le client devrait être impliqué tout au long du projet. L'implication du client permet, entre autres, d'obtenir son appréciation sur les produits intermédiaires et son soutien tout au long du processus de développement. L'auteur vise dans son explication essentiellement le processus de développement, mais l'implication du client peut se manifester également dans l'approvisionnement, l'acquisition et la maintenance. La proposition no. 22 n'est pas déduite ni en contradiction avec d'autres, elle est donc retenue Par contre, la proposition no.5 est déduite de celle-ci tel que présenté à la figure 35.

22 : Involve the customer
5 : Communicate with customers/users

Figure 35 Proposition déduite de la proposition no.22

L'implication du client est aussi liée aux processus de validation et de vérification de la norme ISO/IEC12207 tel que présenté au tableau CXLIX.

Tableau CXLIX

Proposition no.22 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.1 Acquistion	5.1.1 Initiation 5.1.2 Request for proposal preparation 5.1.3 Contract preparation 5.1.4 Supplier monitoring 5.1.5 Acceptance and completion
5.2 Approvisionnement	5.2.4 Planning 5.2.4.5 j)
5.3 Développement	5.3.2 System requirement analysis 5.3.3 System architectural design 5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.10 System integration 5.3.11 System qualification testing 5.3.12 Software installation 5.3.13 Software acceptance support
5.5 Maintenance	5.5.2 Problem and modification analysis 5.5.4 Maintenance review/acceptance
Soutien	
6.4 Verification	6.4.2.1 Contract verification 6.4.2.2 Process verification 6.4.2.3 Requirements verification 6.4.2.4 Design verification 6.4.2.6 Integration verification 6.4.2.7 Documentation verification
6.5 Validation	6.5.2 Validation
6.6 Joint review process	6.6.2 Project management review

Proposition no.23: *Keep design under intellectual control*

Davis (1995) souligne que le design doit être fait et documenté de façon à ce que les développeurs puissent le comprendre dans son ensemble. Il souligne également que le design devrait être construit d'une façon hiérarchique et à l'aide de vues multiples.

La proposition peut être déduite de la proposition no. 20 qui est plus générale : « *Implement a disciplined approach and improve it continuously* ». Cependant, la proposition no. 23 ajoute une précision portant sur l'activité du design. La proposition est donc retenue.

La proposition no. 23 est liée aux processus primaires de développement de la norme ISO/IEC12207, ainsi qu'aux processus de soutien de la gestion de configuration tel que présenté au tableau CL.

Tableau CL

Proposition no.23 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.3 System architectural design 5.3.5 Software architectural design 5.3.6 Software detailed design
Soutien 6.2 Configuration management	

Proposition no.24: *Maintain clear accountability for results*

Boehm (1983) souligne que les individus qui composent les équipes de développement doivent savoir avec précision quels sont les résultats attendus dont ils sont responsables. Cette proposition touche directement l'organisation du travail et le découpage des tâches et des responsabilités. La proposition peut être déduite de la proposition no. 20 qui prône une approche disciplinée. De plus, la proposition no. 1 (*Align incentives for developer*

and customer) a un lien avec la proposition no. 24. Les incitatifs ne peuvent être mis en place, ni vérifiés que si les résultats attendus sont bien définis. Malgré que la proposition no. 24 puisse se déduire de la proposition no. 20, nous considérons qu'elle précise un aspect supplémentaire permettant de mettre en application une approche disciplinée. Elle est donc retenue.

La proposition no. 24 est liée aux processus de type organisationnel de la norme ISO/IEC12207, en particulier aux processus de management tel que présenté au tableau CLI.

Tableau CLI

Proposition no.24 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Organisationnel 7.1 Management	7.1.2 Planning 7.1.2.1 a), d) et e)

Proposition no.25: *Produce software in a stepwise fashion*

Bourque et al. (2002) n'ont pas commenté cette proposition, nous l'interprétons donc comme suit: le logiciel doit être développé à l'aide d'étapes bien définies. Cette proposition peut être déduite de la proposition no. 20 qui est plus générale. Réaliser le logiciel en suivant des étapes s'inscrit dans la portée de la proposition no. 20 à l'effet qu'il faut implémenter une approche disciplinée pour le développement du logiciel. Malgré la déduction possible, la proposition no. 25 ajoute une précision (les étapes) par rapport à la proposition no. 20. La proposition est donc retenue.

La proposition no. 25 est liée aux processus de type primaire de développement de la norme ISO/IEC12207 tel que présenté au tableau CLII.

Tableau CLII

Proposition no.25 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.2 System requirement analysis 5.3.3 System architectural design 5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.7 Software coding and testing 5.3.8 Software integration 5.3.9 Software qualification testing 5.3.10 System integration 5.3.11 System qualification testing 5.3.12 Software installation

Proposition no.26: *Quality is the top priority; long term productivity is a natural consequence of high quality*

Wiegers (1996) affirme que la qualité serait la plus haute priorité dans les activités du génie logiciel. La portée de l'explication de l'auteur se divise en deux principaux points : l'attitude des individus envers la qualité et la qualité des processus et des pratiques. Au niveau des processus, Wiegers souligne que la qualité dépend des processus utilisés pour détecter les erreurs en cours de développement et des pratiques appliquées pour les détecter.

La proposition no. 26 n'est pas déduite, ni en contradiction avec d'autres propositions. Elle est donc retenue. La proposition est liée aux trois groupes principaux de processus de la norme ISO/IEC12207, tel que présenté au tableau CLIII.

Tableau CLIII

Proposition no.26 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	 5.3.2 System requirement analysis 5.3.3 System architectural design 5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.7 Software coding and testing 5.3.8 Software integration 5.3.9 Software qualification testing 5.3.10 System integration 5.3.11 System qualification testing
5.5 Maintenance	5.5.3 Modification implementation
Soutien 6.3 Quality Assusrance 6.4 Verification 6.5 Validation 6.6 Joint review 6.7 Audit	
Organisationnel 7.1 Management 7.3 Processus improvement	 7.1.2.1 g)

Proposition no.27: *Rotate people through product assurance*

Davis (1995) déplore que certaines organisations déplacent des développeurs peu performants vers le groupe d'assurance qualité. Il souligne que ce groupe requiert le même niveau de qualité et de discipline que l'on doit retrouver dans l'équipe de développement. Davis propose plutôt que les *meilleurs* développeurs fassent une visite de quelques mois au sein du groupe d'assurance qualité. La venue des meilleurs

développeurs aura un effet positif sur les activités de ce groupe en permettant de partager les expériences.

Nous constatons que l'idée principale de l'auteur ne se manifeste pas dans la formulation de la proposition. La proposition ne fait pas mention de faire une rotation des meilleures personnes. La proposition satisfait aux deux critères, mais elle comporte une omission importante au niveau de sa formulation.

La proposition est liée aux processus de type organisationnel de la norme ISO/IEC12207 tel que présenté au tableau CLIV.

Tableau CLIV

Proposition no.27 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Organisationnel 7.1 Management	7.1.2 Planning 7.1.2.1 c), d) et e)

Proposition no.28: *Since change is inherent to software, plan for it and manage it*

Bourque et al (2002) ne commentent cette proposition. Nous l'interprétons donc de la façon suivante : le changement fait partie de la réalité du développement du logiciel, ainsi, il faut le planifier et le gérer correctement. Nous considérons que le concept du changement souligné dans la formulation est en lien spécifique au logiciel. Les changements apportés aux ressources humaines, aux délais de livraison ou au budget ne sont pas considérés dans la portée de cette proposition. Nous avons déjà établi que la proposition no. 9 est déduite de la proposition no. 28 tel que présenté à la figure 36.

28 : Since change is inherent to software, plan for it and manage it
9 : Design for maintenance
7 : Design for change

Figure 36 Propositions déduites de la proposition no.22

La proposition satisfait aux deux critères et elle est donc retenue.

La proposition no. 28 est liée à plusieurs processus de la norme ISO/IEC12207 tel que présenté au tableau CLV.

Tableau CLV

Proposition no.28 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Developpement	5.3.1.2 Configuration management (Change control)
5.5 Maintenance	
Soutien	
6.2 Configuration management	6.2.3.1 Change control
Organisationnel	
7.1 Management	7.1.3 Execution and Control

Proposition no.29: *Since tradeoffs are inherent to software engineering, make them explicit and document it*

Bourque et al. (2002) ne fournissent pas d'explication sur la portée de cette proposition. Nous l'interprétons donc comme suit: tous les compromis faits au cours du processus de

développement devraient être explicitement documentés. La proposition comporte un élément implicite de discipline. La proposition peut se déduire de la proposition no. 20. Cependant, la proposition no. 29 précise l'action de documenter les compromis. La proposition satisfait aux deux critères et elle est donc retenue.

La proposition est liée aux processus de type primaire de développement et de maintenance de la norme ISO/IEC12207 tel que présenté au tableau CLVI.

Tableau CLVI

Proposition no.29 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Developpement	5.3.2 System requirement analysis 5.3.3 System architectural design 5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.7 Software coding and testing
5.5 Maintenance	5.5.2 Problem and modification analysis 5.5.3 Modification implementation

Proposition no.30: *Strive to have a peer, rather than a customer, find a defect*

Wiegers (1996) souligne qu'il est impératif que les défauts du logiciel soient détectés à l'interne plutôt que par le client. L'auteur favorise fortement les activités de revue et d'inspection par les pairs. Il souligne également que le développeur peut trouver lui même quelques erreurs, mais les pairs peuvent en trouver un plus grand nombre. La figure 37 présente les propositions déduites de la proposition no. 30.

30 : Strive to have a peer, rather than a customer, find a defect

12 : Don't test your own software

14 : Don't write your own test plans

Figure 37 Propositions déduites de la proposition no.30

Deux propositions (12 et 14) peuvent être déduites de la proposition no. 30. La proposition no. 30 n'est pas déduite ni en contradiction avec d'autres. La proposition est retenue.

La proposition no. 30 est liée aux processus de type primaire et de soutien de la norme ISO/IEC12207 tel que présenté au tableau CLVII.

Tableau CLVII

Proposition no.30 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Developpement	5.3.7 Software coding and testing 5.3.8 Software integration 5.3.9 Software qualification testing 5.3.10 System integration 5.3.11 System qualification testing
5.5 Maintenance	5.5.3 Modification implementation
Soutien	
6.6 Joint review	6.6.3 Technical review

Proposition no.31: *Tailor cost estimation methods*

Davis (1995) souligne essentiellement par cette proposition que les méthodes d'estimation (ex : COCOMO) doivent être adaptées au contexte de l'organisation et des projets afin qu'elles puissent générer de meilleurs résultats. Davis classe cette proposition dans la catégorie management.

La proposition de Davis pourrait être reformulée afin de retirer l'aspect « *cost estimation* » afin d'obtenir une formulation plus générale pouvant s'appliquer à toutes les méthodes. Dans sa formulation actuelle, la proposition a une portée limitée. La proposition n'est pas déduite et ni en contradiction avec d'autres. Elle est donc retenue.

La proposition no. 31 est liée aux processus organisationnel de management, en particulier à l'activité de planification dans laquelle les sous-activités b et h sont spécifiques à l'estimation des coûts.

Tableau CLVIII

Proposition no.31 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Organisationnel 7.1 Management	7.1.2 Planning

Proposition no.32: *To improve design, study previous solutions to similar problems*

Bourque et al. (2002) ne commentent pas la proposition. Nous l'interprétons à l'effet que l'étude de solutions logicielles existantes peut améliorer la conception d'une nouvelle solution au même type de problème. La proposition n'est pas déduite ni en contradiction avec d'autres propositions. Elle est donc retenue.

La proposition no. 32 a une portée essentiellement sur la conception du logiciel. Elle est liée aux processus de type primaire de développement tel que présenté au tableau CLIX.

Tableau CLIX

Proposition no.32 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.3 Développement	5.3.3 System architectural design 5.3.5 Software architectural design 5.3.6 Software detailed design

Proposition no.33: *Use better and fewer people*

Boehm (1983) souligne qu'il y a une grande variance de productivité entre les développeurs. Il suggère donc par cette proposition d'utiliser les meilleurs développeurs plutôt que plusieurs développeurs moyens. Il ajoute qu'un grand nombre de personnes dans un projet augmente la lourdeur des interactions. Boehm suggère donc de retirer de l'équipe de développement les individus les moins performants. La proposition n'est pas déduite ni en contradiction avec d'autres. Elle est donc retenue.

La proposition no. 33 est liée aux processus de type organisationnel de la norme ISO/IEC 12207 tel que présenté au tableau CLX.

Tableau CLX

Proposition no.33 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Organisationnel 7.1 Management	7.1.2 Planning 7.1.2.1 c), d) et e)

Proposition no.35: *Write programs for people first*

La proposition no. 35 a déjà été commentée au niveau de la catégorie produit. Concernant la catégorie processus, la proposition contient le terme « *write* » qui a un lien direct avec le processus puisque c'est celui-ci qui réalise l'écriture des programmes Cette proposition satisfait les deux critères et elle est donc retenue. La proposition no. 35 pourrait être déduite à la proposition no. 9 : « Design for maintenance ». Cependant, la proposition no. 35 se concentre sur l'écriture même des programmes afin qu'ils soient lisibles facilement par d'autres programmeurs.

La proposition no. 35 est lié aux processus de type primaire de développement et de maintenance de la norme ISO/IEC 12207 tel que présenté au tableau CLXI.

Tableau CLXI

Proposition no.35 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Developpement	5.3.7 Software coding and testing
5.5 Maintenance	5.5.3 Modification implementation

Proposition no.37: *Select tests based on the likelihood that they will find faults*

Cette proposition de Davis (1995) a été reformulée à la phase 2 afin de refléter une formulation prescriptive guidant l'action. L'auteur souligne que les cas de tests doivent être sélectionnés de façon à provoquer des erreurs du logiciel. La proposition n'est pas déduite, ni en contradiction avec d'autres. Elle est donc retenue.

La proposition no.37 est lié aux processus de type primaire de développement et de maintenance de la norme ISO/IEC 12207 tel que présenté au tableau CLXII.

Tableau CLXII

Proposition no.37 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire	
5.3 Developpement	5.3.7 Software coding and testing 5.3.8 Software integration 5.3.10 System integration
5.5 Maintenance	5.5.3 Modification implementation (5.5.3.2 a)

Proposition no.38: *Choose a programming language to assure maintainability*

Cette proposition de Davis (1995) a été reformulée à la phase 2 afin de lui donner une forme prescriptive. Davis affirme que le choix du langage de programmation a un impact sur l'effort de maintenance. La proposition n'est pas déduite, ni en contradiction avec d'autres. Elle est donc retenue.

La norme ISO/IEC 12207 fait abstraction du choix du langage de programmation L'annexe E, paragraphe 11, souligne que "*the standard is flexible and usable with[..] any programming languages*". Le langage de programmation peut être aussi considéré comme un outil de développement. À ce titre, la proposition peut être liée aux processus de type organisationnel d'infrastructure qui comprend les outils. Le tableau CLXIII présente la relation entre la proposition no. 38 et la norme ISO/IEC12207.

Tableau CLXIII

Proposition no.38 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Organisationnel	
7.2 Infrastructure	7.2.2 Establishment of the infrastructure

Proposition no.39: *In face of unstructured code, rethink the module and redesign it from scratch*

Cette proposition a également été reformulée à la phase 2. Davis (1995) affirme qu'à l'étape de maintenance, il ne serait pas utile de restructurer le code d'un programme. L'auteur suggère plutôt de repenser le programme et de le réécrire.

La proposition peut se déduire de la proposition no. 6 : *design for maintenance.* Également, la proposition a un lien avec la proposition no. 9 à l'effet de bien définir les artéfacts du logiciel, dont le programme fait partie. Malgré la déduction possible, nous considérons que la proposition no. 39 ajoute une précision guidant l'action du mainteneur. Elle est donc retenue. La figure 38 présente les déductions possibles.

6 : Design for maintenance

9 : In face of unstructured code, rethink the module and redesign it from scratch

Figure 38 Proposition déduite de la proposition no.6

La proposition no. 39 est liée aux processus de type primaire de maintenance tel que présenté au tableau CLXIV.

Tableau CLXIV

Proposition no.39 : relation avec les processus de la norme ISO/IEC12207

Processus	**Activités**
Primaire 5.5 Maintenance	5.5.3 Modification implementation

5.6.3.1 Sommaire de la catégorie processus

Nous avions 36 propositions au départ dans la catégorie processus et 31 ont été conservées. Nous constatons que sur les 36 propositions de départ, quatorze peuvent être déduites de propositions plus générales. Sur ces quatorze propositions, dix apportent une précision supplémentaire par rapport à la proposition parent qui permet de mieux guider l'action. Ces dix propositions ont été conservées et celles-ci sont identifiées par un « * » dans le tableau CLXV qui présente toute les propositions retenues pour la catégorie processus.

Tableau CLXV

Propositions retenues de la catégorie processus

No.	**Proposition**
1	Align incentives for developer and customer
2	Apply and use quantitative measurements in decision making
3	Build software so that it needs a short user manual
4	Build with and for reuse
6	Define software artifacts rigorously
9	Design for maintenance*
10	Determine requirements now*
13	Don't try to retrofit quality
14	Don't write your own test plans*
15	Establish a software process that provides flexibility
16	Fix requirements specification error now*
17	Give product to customers early*
19	Grow systems incrementally
20	Implement a disciplined approach and improve it continuously
21	Invest in the understanding of the problem
22	Involve the customer
23	Keep design under intellectual control*
24	Maintain clear accountability for results*
25	Produce software in a stepwise fashion*
26	Quality is the top priority; long term productivity is a natural consequence of high quality
27	Rotate (high performer) people through product assurance
28	Since change is inherent to software, plan for it and manage it

Tableau CLXV (suite)

No.	Proposition
29	Since tradeoffs are inherent to software engineering, make them explicit and document it*
30	Strive to have a peer, rather than a customer, find a defect
31	Tailor cost estimation methods
32	To improve design, study previous solutions to similar problems
33	Use better and fewer people
35	Write programs for people first
37	Select tests based on the likelihood that they will find faults
38	Choose a programming language according to maintainability
39	In face of unstructured code, rethink the module and redesign it from scratch.*

* : proposition déduite, mais conservée

5.7 Vérification du degré de couverture des propositions par rapport à la norme ISO/IEC12207

Lors de l'analyse des propositions de la catégorie processus, nous avons fait des liens entre chacune des propositions et les groupes de processus identifiés par la norme ISO/IEC12207. Tel que présenté au début de ce chapitre, la classification des processus offerte par la norme permet de vérifier le degré de couverture des propositions retenues. Sur les 31 propositions retenues de la catégorie processus, 29 ont été associés aux processus de la norme ISO/IEC12207. Deux propositions (15 et 20) n'ont pas été associées car elles ne s'appliquent pas à un processus en particulier. Ces deux proposition ont une portée plus générale et possiblement plus fondamentale. Le tableau CLXVI présente la répartition des 29 propositions en fonction des groupes de processus de la norme ISO/IEC12207.

Tableau CLXVI

Répartition des propositions retenues en fonction des groupes de processus de la norme ISO/IEC12207

Processus ISO/IEC12207	**Numéros des propositions associées**
Primaires	
5.1 Acquisition	21, 22
5.2 Approvisionnement	21, 22
5.3 Développement	1, 3, 4, 6, 9, 10, 14, 16, 17, 19, 21, 22, 23, 25, 28, 29, 30, 32, 35, 37
5.4 Exploitation	Aucune
5.5 Maintenance	9, 14, 21, 22, 28, 29, 30, 35, 37, 39
Soutien	
6.1 Documentation	Aucune
6.2 Gestion configuration	23, 28
6.3 Assurance qualité	13, 26
6.4 Vérification	13, 16, 22, 26
6.5 Validation	13, 17, 22, 26
6.6 Revue	22, 26, 30
6.7 Audit	26
6.8 Résolution de problème	21
Organisationnel	
7.1 Management	1, 2, 21, 24, 26, 27, 28, 31, 33
7.2 Infrastructure	38
7.3 Amélioration	26
7.4 Formation	Aucune

À la lumière des résultats présentés par le tableau CLXV, nous constatons que les processus de type primaire recueillent la majorité des propositions, suivis par le groupe de processus de soutien et finalement au troisième rang, le groupe de processus organisationnels. Le tableau CLXVII présente le dénombrement des propositions associées à chacun des trois groupes de processus de la norme ISO/IEC12207. Le nombre total dépasse 29 du fait que certaines propositions se retrouvent à plus d'un endroit.

Tableau CLXVII

Dénombrement des propositions par groupes de processus

Groupes de processus ISO/IEC12207	Nombre de propositions
Primaire (34)	
Développement	20
Maintenance	10
Acquisition	2
Approvisionnement	2
Exploitation	0
Soutien (17)	
Vérification	4
Validation	4
Revue	3
Gestion configuration	2
Assurance qualité	2
Audit	1
Résolution de problème	1
Documentation	0
Organisationnel (11)	
Management	9
Amélioration processus	1
Infrastructure	1
Formation	0

5.7.1 Groupe des processus primaires

Nous constatons que 20 des 29 propositions (71%) se retrouvent associées au processus primaire de développement. En deuxième rang, se retrouve le processus primaire de maintenance qui recueille 10 propositions (36%). Nous soulignons que huit des neuf propositions du groupe maintenance sont aussi associées au groupe développement. Seule la proposition no. 39 du groupe maintenance n'est pas partagée.

Les groupes de processus d'acquisition et d'approvisionnement ne recueillent que deux propositions, les deux mêmes pour les deux groupes. Enfin, nous constatons qu'aucune proposition n'est associée au groupe de processus d'exploitation.

À la lumière de ces résultats, nous constatons une certaine tendance à l'effet que les propositions retenues visent principalement les activités de développement et de maintenance. Cependant, cette constatation ne signifie pas qu'il n'existe pas de proposition (éventuellement des principes) associée au groupe exploitation.

5.7.2 Groupe des processus de soutien

Le groupe des processus de soutien se subdivise en huit processus. Les processus de vérification, de validation et de revue recueillent la majorité des propositions. Nous constatons que le processus de documentation ne comporte aucune proposition. Cependant, la proposition no. 3 (Use documentation standards) pourrait y être associée à première vue. En premier lieu, cette proposition a été classée dans la catégorie des produits. L'utilisation de normes améliore la qualité des produits. En deuxième lieu, la proposition ne comporte pas le sens de documenter les activités comme le processus de documentation (ISO/IEC 12207 - 6.1) l'indique. C'est la raison qui fait en sorte que la proposition no. 3 n'est pas liée au processus de documentation

5.7.3 Groupe des processus organisationnels

Ce groupe se place au troisième rang dans la répartition des propositions en recueillant que onze propositions. Le processus de management en recueille à lui seul huit des onze propositions. Les processus d'infrastructure et d'amélioration ne recueillent qu'une proposition, ce qui est bien peu comparativement à l'importance de ces processus. Le processus de formation n'est associé à aucune proposition.

De ces résultats, nous constatons que le processus primaire de développement vient au premier rang avec 20 propositions, suivi du processus primaire de maintenance avec 10 propositions et du processus organisationnel de management avec huit propositions. Globalement, nous constatons qu'après le retrait des doublons, ces trois processus regroupent 25 des 29 propositions retenues. Si nous renons compte que huit des neuf

propositions du groupe maintenance se retrouvent aussi au niveau du groupe développement, nous constatons qu'en fait il y a deux principaux pôles de processus soit: le groupe des processus de développement et le groupe des processus de management. Il y a donc une nette tendance à l'effet que les propositions du groupe processus retenues soutiennent majoritairement les activités de développement, de maintenance et de management. Nous constatons qu'aucun processus de soutien ne se classe parmi les trois premiers rangs.

5.8 Conclusion de la phase 3

Dans ce chapitre, les 39 propositions issues de la phase 2 ont été analysées. Les propositions ont été classées selon les trois catégories identifiées soit : individu, produit et processus. Par la suite, nous avons appliqué les deux critères d'ensemble suivants sur les 39 propositions:

1. Les principes devraient être indépendants (non déduits) (Boehm 1983)
2. Un principe ne doit pas contredire un autre principe connu. (Bourque et al. 2002)

Aucune des 39 propositions ne s'est révélée en contradiction avec une autre proposition. Ce critère n'a donc pas permis d'éliminer des propositions. Au niveau du premier critère, son application a évolué au cours de l'analyse. Nous avons évalué que certaines propositions, malgré qu'elles puissent être déduites, comportaient une précision intéressante qui méritait une attention. Puisqu'un principe guide l'action, certaines propositions précisent plus explicitement une action concrète à faire que la formulation de la proposition parent, souvent plus générale. À la lecture de certaines propositions parents, il peut y avoir interprétation différente des actions à faire. Les propositions déduites qui précisent explicitement l'action à faire sont plus facilement applicables que les propositions plus générales. C'est dans cette optique que nous avons évalué les

propositions déduites. Est-que la proposition ajoute une précision guidant l'action par rapport à la proposition parent? Dans l'affirmative, la proposition a été conservée.

Au niveau de la catégorie individu, nous avions six propositions au départ, mais quatre ont été conservées. Au niveau de la catégorie produit, nous avions sept propositions au départ et toutes ont été conservées même si une de celles-ci est déduite d'une autre, mais comme elle ajoute une précision, elle a été conservée. La catégorie processus regroupe 36 des 39 propositions. Nous avons écarté quatre propositions qui sont déduites, mais sans ajouter de précisions supplémentaires. Nous avons conservé dix propositions qui peuvent être déduites, mais qui ajoutent une précision guidant l'action à entreprendre par rapport à la proposition parent. La proposition no.8 a été écartée compte tenu de sa formulation déficiente et l'explication contradictoire de l'auteur. Sur les 39 propositions au départ, 34 propositions ont été retenues, tel que présenté au tableau CLXVIII.

Tableau CLXVIII

Les 34 propositions retenues suite à la phase 3

No.	**Proposition**
1	Align incentives for developer and customer
2	Apply and use quantitative measurements in decision making
3	Build software so that it needs a short user manual
4	Build with and for reuse
6	Define software artifacts rigorously
9	Design for maintenance*
10	Determine requirements now*
11	Don't overstrain your hardware
13	Don't try to retrofit quality
14	Don't write your own test plans*
15	Establish a software process that provides flexibility
16	Fix requirements specification error now*
17	Give product to customers early*
19	Grow systems incrementally
20	Implement a disciplined approach and improve it continuously
21	Invest in the understanding of the problem

Tableau CLXVIII (suite)

No.	Proposition
22	Involve the customer
23	Keep design under intellectual control*
24	Maintain clear accountability for results*
25	Produce software in a stepwise fashion*
26	Quality is the top priority; long term productivity is a natural consequence of high quality
27	Rotate (high performer) people through product assurance
28	Since change is inherent to software, plan for it and manage it
29	Since tradeoffs are inherent to software engineering, make them explicit and document it*
30	Strive to have a peer, rather than a customer, find a defect
31	Tailor cost estimation methods
32	To improve design, study previous solutions to similar problems
33	Use better and fewer people
34	Use documentation standards
35	Write programs for people first
36	Know software engineering's techniques before using development tools
37	Select tests based on the likelihood that they will find faults
38	Choose a programming language to assure maintainability
39	In face of unstructured code, rethink the module and redesign it from scratch.*

* : propositions déduites, mais conservées

Nous avons effectué une association des propositions de la catégorie processus avec les groupes de processus identifiés par la norme ISO/IEC12207. Nous observons que les propositions s'associent principalement à trois processus: développement, maintenance et management. Certains processus, telle la formation, la documentation et l'exploitation ne sont pas associés à aucune proposition.

CHAPITRE 6

PHASE 4 – ÉVALUATION DU DEGRÉ DE COUVERTURE DES PRINCIPES

La phase 3 a retenu 34 propositions. Les sept critères d'identification ont été appliqués, cinq critères à la deuxième phase et deux critères à la troisième phase. Le processus d'élimination des propositions est maintenant complété.

6.1 Objectif de la phase 4

L'objectif principal de la phase 4 est de procéder à une vérification de la couverture des propositions candidates retenues, en deux volets. Nous désirons, maintenant, vérifier si les propositions retenues soutiennent le génie logiciel. Le premier volet consiste à analyser les propositions en fonction des éléments du modèle d'ingénierie présenté par Moore (2006). Le deuxième volet consiste à vérifier la couverture des propositions en fonction du corpus de normes du génie logiciel de l'IEEE.

6.2 Couverture des propositions en fonction du modèle d'ingénierie.

James W. Moore (2006) présente un modèle type de l'ingénierie représentant les concepts des activités de génie (figure 39). Dans ce volet, nous analysons chacune des propositions retenues en fonction des éléments de ce modèle. L'objectif de cette étape est de vérifier si tous les éléments du modèle sont couverts par les propositions. Nous faisons l'hypothèse qu'ayant retenu des propositions pertinentes au génie logiciel, nous devrions obtenir une couverture de l'ensemble des éléments du modèle.

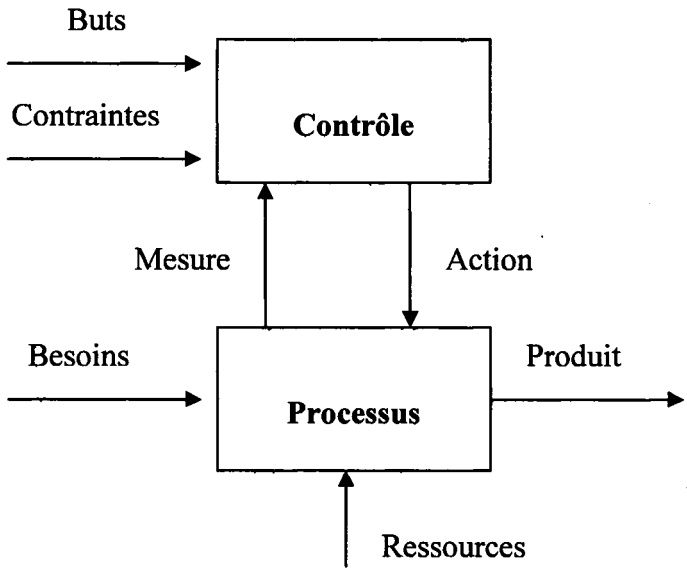

Figure 39 Modèle de l'ingénierie (Traduit de Moore 2006)

6.2.1 Méthode de recherche pour la phase 4

Dans un premier temps, les éléments du modèle sont décrits afin de bien en saisir la portée. Par la suite, pour chacune des propositions, un bref rappel est fait concernant sa signification lorsque donnée par l'auteur. Les éléments ou le sens explicite de la proposition est identifié et une association est faite avec les éléments du modèle.

Une proposition peut aussi contenir des éléments implicites, par exemple, en termes de conséquences, sans qu'ils soient explicitement formulés dans la proposition. Ces éléments implicites contenus dans la proposition sont aussi identifiés.

Une vérification est faite avec le classement des propositions selon les trois catégories (produit, processus, individu) réalisé antérieurement afin d'assurer une certaine cohérence. Une synthèse sera faite sur les observations constatées.

6.2.2 Description des éléments du modèle

Le modèle est composé de deux principaux pôles : le processus et le contrôle. Le processus représente l'ensemble des activités à réaliser pour créer le produit, le logiciel. Le contrôle représente le volet management et ingénierie qui s'assure que les activités du processus se déroulent dans le respect des buts et des contraintes du projet et de l'entreprise. Le modèle peut s'appliquer globalement au processus d'ingénierie ou à chacun de ses sous-processus. Décrivons maintenant chacun des éléments du modèle.

6.2.2.1 Processus

Le processus reçoit en entrée les besoins exprimés par le client. Une suite d'activités s'en suit pour concevoir et réaliser un produit répondant aux besoins exprimés. Le processus requiert des ressources pour réaliser les activités. Concernant le génie logiciel, le processus intègre les activités principales suivantes :

- Les exigences logicielles
- La conception
- La construction
- Les tests
- La maintenance

Ces activités correspondent aux cinq premiers domaines de connaissance de SWEBOK (2004). Il est à noter que SWEBOK intègre aussi certains aspects de mesure et de contrôle à ces domaines de connaissance.

6.2.2.2 Besoins

Le processus de réalisation du produit requiert les besoins du client. Les besoins représentent, entre autres, certaines caractéristiques que le produit final doit avoir pour satisfaire les attentes du client. Pour le génie logiciel, les besoins peuvent être les exigences fonctionnelles et non-fonctionnelles exprimées par le client. Le processus aura

comme activité première de faire la synthèse des besoins exprimés et par la suite, de les transformer en spécifications.

6.2.2.3 Ressources

Le déroulement du processus requiert des ressources. Ces ressources sont, entre autres :

- La main d'œuvre
- Les matériaux (moins applicable au génie logiciel)
- Les outils (logiciel et matériel)
- Les normes
- Les méthodes
- Les livres de référence
- Les produits intermédiaires

Il est à noter qu'un produit intermédiaire issu d'un sous-processus est considéré, lui-même, comme une ressource lorsqu'il est utilisé par un autre sous-processus.

6.2.2.4 Produit

Globalement, le produit final est l'objectif ou le résultat attendu du processus du génie. Dans le cas du génie logiciel, le logiciel et le guide d'utilisation sont les produits finaux issus du processus de développement. Le processus de développement se découpe en sous-processus. Ceux-ci génèrent un ou plusieurs produits intermédiaires qui seront utilisés comme des ressources par les sous-processus suivants.

6.2.2.5 Contrôle

Le contrôle est le pôle orienté management et ingénierie où des décisions sont prises pour gérer le processus, le corriger et l'améliorer. Le contrôle s'assure que le processus se déroule normalement afin que les objectifs (les buts) soient atteints dans le respect des

contraintes. Le contrôle doit s'assurer que le projet se déroule conformément aux plans, aux budgets et aux échéanciers prévus. Le contrôle permet aussi d'améliorer le processus afin, entre autres, d'augmenter la productivité ou la qualité du produit. Afin de jouer pleinement son rôle, le contrôle doit s'alimenter en information provenant de mesures prises au niveau du processus. L'analyse des mesures se fait en conjonction avec les buts et contraintes afin de détecter des écarts. Si les écarts nécessitent un ajustement, le contrôle déclenchera une action corrective vers le processus.

6.2.2.6 Mesures

Les mesures sont les données (nominale, ordinale, intervalle, ratio, absolue) recueillies au niveau du processus ou des produits. Le contrôle procède à l'analyse des données recueillies en considérant les buts et les contraintes et ceci pour détecter des écarts. Si des écarts sont constatés, le contrôle va générer une action vers le processus afin de diminuer ces écarts.

6.2.2.7 Action

L'action est une intervention issue du contrôle afin de diriger le processus selon les plans prévus ou de le rectifier suite à la mesure d'un écart. Le contrôle, par la prise de nouvelles mesures, pourra vérifier si l'action a porté fruit.

6.2.2.8 Buts

Les buts sont principalement des objectifs d'affaires de l'entreprise. Les objectifs suivants sont, entre autres, des exemples de buts qui influencent le contrôle et implicitement le processus d'ingénierie.

- Réalisation de profits, rentabilité de l'entreprise
- Production à faible coûts
    - Favoriser la réutilisation

- Faciliter l'effort de maintenance
- Préserver ou accroitre la part de marché
- Satisfaction des clients
- Qualité des produits

6.2.2.9 Contraintes

Le contrôle doit aussi tenir compte de contraintes contextuelles au projet. Les budgets alloués au projet, les délais de livraison, l'arrimage du projet à un autre projet sont, entre autres, des contraintes contextuelles.

6.2.3 Principes et modèle

Dans cette section, nous analysons les 34 propositions retenues en fonction des éléments du modèle de l'ingénierie présenté par Moore (2006).

Proposition no.1: Align incentives for developer and customer

L'explication fournie par Davis (1995) vise essentiellement les développeurs. Ceux-ci doivent être motivés et responsabilisés dans l'atteinte des objectifs, tels le respect des échéanciers. Davis souligne que le management devrait mettre en place des mesures positives et négatives pour l'atteinte ou non des objectifs. La proposition est étroitement liée à la proposition no. 24 (*maintain clear accountability of results*). La mise en place d'incitatifs ne peut se faire que si les résultats et les responsabilités de chacun sont clairement définies et communiqués.

Le sens explicite de la proposition souligne que le processus doit intégrer des incitatifs pour les développeurs afin d'atteindre les résultats prévus. D'une façon implicite, le management veut atteindre ses buts et éviter, à titre d'exemple, les dépassements des

échéanciers et des budgets (conséquences). Le contrôle doit intervenir pour vérifier (à l'aide de mesures) si les objectifs sont atteints ou non. Le contrôle, par l'action, soulignera l'atteinte des objectifs (par des récompenses) ou les écarts (réprimandes). Les incitatifs peuvent être vus comme des contraintes sur les individus (ressources)

Tableau CLXIX

Éléments d'ingénierie impliqués pour la proposition no.1

Align incentives for developer and customer	
Sens explicite	Sens implicite
• Processus	• Contrôle • Action • Mesure • Contrainte • Ressource • But

Proposition no.2: Apply and use quantitative measurements in decision making

Le contrôle doit s'alimenter en information en provenance du processus à l'aide des mesures. La proposition souligne explicitement d'appliquer et d'utiliser les mesures dans la prise de décisions qui est le cœur du contrôle. La proposition est associée à l'élément mesure ainsi qu'à l'élément contrôle du modèle où se fait la prise de décision basée, entre autres, sur l'analyse des mesures. D'une façon implicite, il y aura une action de générée par la prise de décision.

Tableau CLXX

Éléments d'ingénierie impliqués pour la proposition 2

Apply and use quantitative measurements in decision making	
Sens explicite	Sens implicite
• Mesure • Contrôle	• Action

Proposition no.3: Build software so that it needs a short user manual

Cette proposition concerne le produit final, le logiciel. Le thème de la facilité d'utilisation est central à cette proposition, même si implicite. Si le logiciel est simple à utiliser, le recours à des explications supplémentaires du manuel d'utilisation ne sera pas requis. Le terme « build » implique explicitement le processus. Également, la proposition souligne une contrainte explicite à l'effet que le manuel d'utilisation doit être court. De plus, la proposition mentionne explicitement les deux produits finaux du processus, le logiciel et le guide d'utilisation. La facilité d'utilisation peut être aussi vue, implicitement, comme un besoin exprimé par le client.

Tableau CLXXI

Éléments d'ingénierie impliqués pour la proposition no.3

Build software so that it needs a short user manual	
Sens explicite	**Sens implicite**
• Processus • Produit • Contrainte	• Besoin

Proposition no.4: Build with and for reuse

La proposition comporte deux aspects. Le premier concerne le « build with » qui est explicitement une contrainte imposée sur le processus afin de construire le logiciel avec des composants déjà faits. Le deuxième aspect est le « for reuse » qui est une contrainte sur les produits réalisés. Ces produits doivent être conçus pour être réutilisables et agir implicitement comme ressources (implicitement) à un autre projet. D'une façon implicite, la réutilisation peut favoriser une réduction des coûts et la réduction des délais de développement. La réutilisation peut être un but du management et des architectes.

Tableau CLXXII

Éléments d'ingénierie impliqués pour la proposition no.4

Build with and for reuse	
Sens explicite	Sens implicite
• Contrainte • Produit • Processus	• But • Ressource

Proposition no.6: Define software artifacts rigorously

La proposition souligne d'une façon explicite par le qualificatif « rigorously » que chacun des produits intermédiaires (artefact) doit être défini avec rigueur. Les produits intermédiaires sont définis par le processus, mais l'aspect de la rigueur sera imposé et vérifié par le volet contrôle. D'une façon implicite, cette proposition souligne qu'en définissant avec rigueur les produits intermédiaires, des imprécisions pourraient être corrigées, des oublis pourraient être détectés. Implicitement, des conséquences sur les coûts et les délais pourraient être diminuées (but).

Tableau CLXXIII

Éléments d'ingénierie impliqués pour la proposition no.6

Define software artifacts rigorously	
Sens explicite	Sens implicite
• Processus • Produit • Contrôle	• But • Mesure • Action

Proposition no.9: Design for maintenance

Les activités de maintenance peuvent être coûteuses si le logiciel a été développé sans avoir tenu compte de sa facilité future à être modifié. Pour le management, la facilité

d'entretien du logiciel peut être un but. Celui-ci permettra de réaliser des modifications au logiciel dans des délais plus courts et ainsi réduire les coûts de l'activité de maintenance. La proposition souligne explicitement que le processus de design doit tenir compte des aspects de la maintenance, c'est une forme de contrainte. D'une façon implicite, la proposition touche les ressources, telles les normes, les gabarits et autres, qui permettent, entre autres, d'uniformiser le style des produits et en faciliter la maintenance par la suite. La facilité de maintenance est aussi une caractéristique intrinsèque du produit.

Tableau CLXXIV

Éléments d'ingénierie impliqués pour la proposition no.9

Design for maintenance	
Sens explicite	Sens implicite
• Processus • But • Contrainte	• Produit • Ressource

Proposition no.10: Determine requirements now

La proposition souligne explicitement de déterminer les exigences et ce « maintenant ». Le premier volet s'associe avec l'élément processus du modèle, tandis que le volet « maintenant » est explicitement une contrainte. D'une façon implicite, la contrainte implique le contrôle afin de vérifier si les exigences sont bien définies à l'étape de la spécification des exigences. Laisser en suspens des exigences peut être considéré comme une incertitude et un risque supplémentaire sur le projet, ce que le contrôle veut éliminer ou réduire.

Tableau CLXXV

Éléments d'ingénierie impliqués pour la proposition no.10

Determine requirements now	
Sens explicite	Sens implicite
• Processus • Contrainte	• Contrôle

Proposition no.11: Don't overstrain your hardware

Davis (1995) souligne que la surcharge du matériel peut entrainer des modifications importantes et coûteuses au logiciel, une fois complété. La proposition souligne explicitement une contrainte sur le produit final. Les modifications à faire afin de rendre le logiciel moins gourmand en ressources matérielles peuvent être coûteuses, retarder sa livraison et même être la cause d'un échec. D'une façon implicite, la proposition ne mentionne pas le produit, mais il y un lien avec le produit final, puisqu'il est en cause dans la surcharge du matériel. De plus, le contrôle est impliqué afin de s'assurer que le logiciel développé tourne adéquatement sur le matériel prévu.

Tableau CLXXVI

Éléments d'ingénierie impliqués pour la proposition no.11

Don't overstrain your hardware	
Sens explicite	Sens implicite
• Contrainte	• Produit • Contrôle

Proposition no.13: Don't try to retrofit quality

La qualité ne peut s'ajouter au logiciel une fois le développement terminé. Le processus doit donc intégrer des activités d'assurance qualité à partir du début du processus jusqu'à

la fin. La proposition est donc explicitement orientée vers le processus. D'une façon implicite, la proposition touche à la qualité des produits intermédiaires et finaux. Le contrôle doit s'assurer que les activités d'assurance qualité soient réalisées au niveau du processus et que des mesures soient prises pour s'en assurer.

Tableau CLXXVII

Éléments d'ingénierie impliqués pour la proposition no.13

Don't try to retrofit quality	
Sens explicite	Sens implicite
• Processus	• Produit • Contrôle • Mesure

Proposition no.14: Don't write your own test plans

Davis (1995) souligne qu'aucun développeur ne devrait écrire les plans de test des composants logiciels qu'il a lui-même développés. La proposition exprime explicitement une contrainte sur le processus. D'une façon implicite, le contrôle doit s'assurer que cette contrainte est suivie au niveau du processus.

Tableau CLXXVIII

Éléments d'ingénierie impliqués pour la proposition no.14

Don't write your own test plans	
Sens explicite	Sens implicite
• Contrainte • Processus	• Contrôle

Proposition no.15: Establish a software process that provides flexibility

Le processus mis en place doit avoir la flexibilité de pouvoir s'adapter au contexte spécifique du projet et de l'organisation. La flexibilité peut être vue comme un objectif du management. La proposition vise explicitement un but à atteindre. Également, la proposition mentionne l'élément processus. Il y a aussi un aspect de contrainte sur le processus afin qu'il s'ajuste au contexte du projet.

Tableau CLXXIX

Éléments d'ingénierie impliqués pour la proposition no.15

Establish a software process that provides flexibility	
Sens explicite	Sens implicite
• But • Processus • Contrainte	

Proposition no.16: Fix requirements specification error now

Davis (1995) souligne que les erreurs au niveau des exigences doivent être corrigées dès l'étape de la spécification des exigences du logiciel et ce, avant même d'entreprendre les étapes subséquentes. Nous identifions explicitement dans cette proposition deux éléments principaux. En premier lieu, le terme « fix » représente une action. Deuxièmement, le terme « now » est une contrainte sur le processus. D'une façon implicite, toute la boucle de contrôle est impliquée. Des mesures doivent être prises pour constater les erreurs dans les spécifications. Par la suite, le contrôle, face à l'écart détecté et à la contrainte (du « now ») doit enclencher une action corrective (« fix ») vers le processus afin de faire corriger les écarts.

Tableau CLXXX

Éléments d'ingénierie impliqués pour la proposition no.16

Fix requirements specification error now	
Sens explicite	Sens implicite
• Action • Contrainte • Processus	• Contrôle • Mesure

Proposition no.17: Give product to customers early

Davis (1995) souligne que le client peut s'impliquer dans le processus en utilisant des versions prototypes du logiciel. Cependant, l'explication de l'auteur ne stipule pas que le processus doit livrer le produit final plus tôt.

D'une façon explicite, la proposition demande au processus de fournir des prototypes au client. Également, la proposition ajoute une contrainte (« early ») au processus. D'une façon implicite, la proposition est liée au contrôle qui doit vérifier si le processus a bien réalisé le prototype et si la contrainte est respectée.

Tableau CLXXXI

Éléments d'ingénierie impliqués pour la proposition no.17

Give product to customers early	
Sens explicite	Sens implicite
• Processus • Contrainte	• Contrôle

Proposition no.19: Grow systems incrementally

Davis (1995) souligne que le développement du logiciel devrait s'effectuer d'une façon incrémentale par l'ajout successif de fonctions. La proposition peut s'associer explicitement au processus. D'une façon implicite, la voie incrémentale permet au management de diminuer les risques du projet, ce qui peut être considéré comme un but. Également, le développement incrémental peut contenir un aspect de contrainte sur le processus.

Tableau CLXXXII

Éléments d'ingénierie impliqués pour la proposition no.19

Grow systems incrementally	
Sens explicite	Sens implicite
• Processus	• But • Contrainte

Proposition no.20: Implement a disciplined approach and improve it continuously

La proposition comporte une contrainte explicite sur le processus : « disciplined » et une autre sur le contrôle : « continuously ». Le processus doit se dérouler d'une façon disciplinée. La proposition souligne, également, qu'il doit y avoir amélioration continue du processus. Cette contrainte positive porte sur le contrôle qui, à l'aide des mesures, procède à des ajustements, sous forme d'actions, au niveau du processus. D'une façon implicite, le contrôle doit effectuer des mesures afin de vérifier si le processus s'exécute de façon disciplinée et, les cas échéant, des actions peuvent s'en suivre. L'amélioration du processus peut être un but du management afin de diminuer les coûts et les délais de développement et d'augmenter la qualité du produit. En bout de ligne, la qualité du produit devrait être influencée par l'amélioration du processus.

Tableau CLXXXIII

Éléments d'ingénierie impliqués pour la proposition no.20

Implement a disciplined approach and improve it continuously	
Sens explicite	Sens implicite
• Contrainte • Processus • Contrôle	• But • Mesure • Action • Produit

Proposition no.21: Invest in the understanding of the problem

Le terme investir souligne qu'il faut déployer les efforts et les ressources nécessaires pour comprendre le problème à résoudre. La proposition est associée à l'élément processus du modèle. Le processus doit comporter l'étape d'élucidation du problème. D'une façon implicite, il y a des conséquences à ne pas élucider correctement le problème. À titre d'exemple, une mauvaise élucidation peut entrainer des dépassements de coûts, de délais et la non satisfaction du client envers le produit final. Le management doit aussi consacrer (sous forme d'action) les ressources nécessaires pour mettre en application cette proposition.

Tableau CLXXXIV

Éléments d'ingénierie impliqués pour la proposition no.21

Invest in the understanding of the problem	
Sens explicite	Sens implicite
• Processus	• Action • Ressource • Contrôle

Proposition no.22: Involve the customer

La proposition souligne que le processus doit impliquer le client. C'est donc une forme de contrainte imposée au processus. Le client est considéré alors comme une ressource pour le processus. D'une façon implicite, le contrôle doit vérifier si le processus a effectivement impliqué le client et ce au moment prévu.

Tableau CLXXXV

Éléments d'ingénierie impliqués pour la proposition no.22

Involve the customer	
Sens explicite	Sens implicite
• Processus • Contrainte • Ressource	• Contrôle

Proposition no.23: Keep design under intellectual control

Davis (1995), dans ses commentaires, limite la portée de cette proposition à l'effet que le design doit être fait et documenté de façon à ce que les développeurs puissent le comprendre dans son ensemble. L'auteur identifie des techniques à utiliser telles la conception hiérarchique et les vues multiples. Selon l'explication fournie par l'auteur la dimension du contrôle intellectuel est plutôt faible. Le sens donné par l'auteur est orienté vers la conception à l'aide des techniques suggérées. La proposition est liée explicitement aux éléments processus et contrôle du modèle. D'une façon implicite, la proposition sous-entend une façon de documenter le design, comme un produit intermédiaire. Également, la mesure pourrait être impliquée afin de contrôler le design et mêmes des actions correctives pourraient être déclenchées.

Tableau CLXXXVI

Éléments d'ingénierie impliqués pour la proposition no.23

Keep design under intellectual control	
Sens explicite	Sens implicite
• Processus • Contrôle	• Produit • Mesure • Action

Proposition no.24: Maintain clear accountability for results

Boehm (1983) souligne que les individus impliqués dans le processus de développement doivent être informés des résultats attendus et de leur responsabilité face à l'atteinte de ceux-ci. Cette proposition est associée au processus et au contrôle. Le processus doit informer les individus des attentes. D'une façon implicite, le contrôle doit s'assurer que le processus informe les individus, donc les ressources.

Tableau CLXXXVII

Éléments d'ingénierie impliqués pour la proposition no.24

Maintain clear accountability for results	
Sens explicite	Sens implicite
• Processus • Contrôle	• Ressources (individus)

Proposition no.25: Produce software in a stepwise fashion

La proposition souligne explicitement que le logiciel doit être développé à l'aide d'un processus composé d'étapes. D'une façon implicite, le contrôle doit s'assurer qu'il en soit ainsi.

Tableau CLXXXVIII

Éléments d'ingénierie impliqués pour la proposition no.25

Produce software in a stepwise fashion	
Sens explicite	Sens implicite
• Processus	• Contrôle

Proposition no.26: Quality is the top priority; long term productivity is a natural consequence of high quality

La proposition est explicitement associée à un but du management où la qualité serait la priorité. De plus, la productivité à long terme est liée directement à l'amélioration du processus. D'une façon implicite, cette proposition implique le produit qui devrait intégrer certaines des caractéristiques de qualité. Wiegers (1996) ajoute également, que l'attitude des individus envers la qualité est cruciale. Le contrôle s'assurera que les activités d'assurance qualité sont faites et que les objectifs de qualité sont atteints.

Tableau CLXXXIX

Éléments d'ingénierie impliqués pour la proposition no.26

Quality is the top priority; long term productivity is a natural consequence of high quality	
Sens explicite	Sens implicite
• But • Processus	• Produit • Ressources (individus) • Contrôle

Proposition no.27: Rotate people through product assurance

Davis (1995) souligne que certaines organisations envoient les développeurs peu performants vers le groupe d'assurance qualité. Davis suggère plutôt que les

organisations mettent en place un processus afin que les meilleurs développeurs fassent un séjour au sein du groupe d'assurance qualité. Ces développeurs pourront par la suite intégrer les aspects de qualité dans le processus de développement et contribuer à l'amélioration de la qualité du processus et en bout de ligne du produit.

La proposition souligne explicitement une contrainte sur le processus de gestion des ressources humaines. De plus, elle souligne explicitement les individus ciblés. D'une façon implicite, le management cherche à améliorer la qualité du processus et la qualité du produit en bout de ligne. Ainsi, c'est un objectif du management.

Tableau CXC

Éléments d'ingénierie impliqués pour la proposition no.27

Rotate (top performer) people through product assurance	
Sens explicite	Sens implicite
• Contrainte • Processus • Ressource	• But • Produit

Proposition no.28: Since change is inherent to software, plan for it and manage it

La proposition débute par une constatation à l'effet que le changement est inévitable. De ce fait, la proposition souligne qu'il faut le planifier et le gérer. La proposition mentionne explicitement un aspect lié au contrôle. De plus, la constatation est une forme de contrainte. D'une façon implicite, une action est générée par le contrôle dans le but de gérer le changement au niveau du processus. La mesure apportera au contrôle les demandes de changement et le contrôle décidera, en fonction des buts et contraintes, lesquelles seront effectuées.

Tableau CXCI

Éléments d'ingénierie impliqués pour la proposition no.28

Since change is inherent to software, plan for it and manage it	
Sens explicite	Sens implicite
• Contrôle • Contrainte	• Action • Mesure

Proposition no. 29: Since tradeoffs are inherent to software engineering, make them explicit and document it

Le processus de développement doit documenter explicitement tous les compromis faits. La proposition est associée explicitement au processus et une exerce une forme de contrainte sur celui-ci. D'une façon implicite, le contrôle peut vérifier si le processus documente dans les faits les compromis.

Tableau CXCII

Éléments d'ingénierie impliqués pour la proposition no.29

Since tradeoffs are inherent to software engineering, make them explicit and document it	
Sens explicite	Sens implicite
• Processus • Contrainte	• Contrôle

Proposition no.30: Strive to have a peer, rather than a customer, find a defect

Wiegers (1996) souligne que les défauts du logiciel devraient être détectés à l'interne par des collègues, plutôt que par le client. La proposition souligne explicitement une contrainte sur le processus du test du logiciel. D'une façon implicite, le contrôle doit

s'assurer de la qualité du processus de test du logiciel et vérifier par des mesures le taux de défauts trouvés par le client suite à l'installation du logiciel.

Tableau CXCIII

Éléments d'ingénierie impliqués pour la proposition no.30

Strive to have a peer, rather than a customer, find a defect	
Sens explicite	Sens implicite
• Contrainte • Processus	• Contrôle

Proposition no.31: Tailor cost estimation methods

Davis (1995) souligne que les méthodes d'estimation doivent être adaptées au contexte propre de l'organisation et des projets. La proposition a une portée strictement de niveau management. Elle est donc associée à l'élément contrôle. D'une façon implicite, les méthodes d'estimation s'alimentent de mesures.

Tableau CXCIV

Éléments d'ingénierie impliqués pour la proposition no.31

Tailor cost estimation methods	
Sens explicite	Sens implicite
• Contrôle	• Mesures

Proposition no.32: To improve design, study previous solutions to similar problems

La proposition souligne que le processus de design tienne compte de solutions antérieures à un problème similaire dans le but d'améliorer la conception. La proposition est explicitement associée à l'élément processus du modèle. D'une façon implicite,

l'application de la proposition peut faire en sorte de diminuer les temps de développement, d'éviter des erreurs dans le design, de profiter des expériences antérieures et ainsi économiser des ressources. C'est implicitement un but que le management pourrait avoir.

Tableau CXCV

Éléments d'ingénierie impliqués pour la proposition no.32

To improve design, study previous solutions to similar problems	
Sens explicite	Sens implicite
• Processus • But	• But (conséquences)

Proposition no.33: Use better and fewer people

Boehm (1983) souligne qu'il y a une grande variance de productivité entre les développeurs, ainsi il suggère d'utiliser les meilleurs développeurs plutôt que plusieurs développeurs de calibre moyen. La proposition souligne explicitement une contrainte sur le management du projet (le contrôle). D'une façon implicite, il y a des motivations (buts) à appliquer cette proposition afin d'accélérer le processus, de diminuer les coûts et d'améliorer la qualité. La proposition a un lien avec l'élément ressources du modèle puisqu'il est question des ressources humaines.

Tableau CXCVI

Éléments d'ingénierie impliqués pour la proposition no.33

Use better and fewer people	
Sens explicite	Sens implicite
• Contrainte • Contrôle • Ressources	• But

Proposition no.34: Use documentation standards

Davis (1995) souligne que les normes offrent une structure permettant d'organiser clairement les activités. Les produits intermédiaires sont couverts par les normes telles l'IEEE. Les normes et standards sont considérés comme des ressources utilisées par le processus. D'une façon implicite, l'utilisation de normes peut être aussi vue comme une contrainte du management sur le processus.

Tableau CXCVII

Éléments d'ingénierie impliqués pour la proposition no.34

Use documentation standards	
Sens explicite	Sens implicite
• Ressources	• Contrainte • Produit • Processus

Proposition no.35: Write programs for people first

Davis (1995) souligne que les programmes n'ont plus à être écrits de façon cryptique à cause du coût du matériel. L'auteur souligne que la ressource coûteuse est maintenant la main d'œuvre. Le développeur (et aussi l'organisation) doit garder à l'esprit que le programme est écrit une fois, mais lu et modifié plusieurs fois au cours de sa vie utile. La proposition peut se diviser en deux volets. En premier lieu, « *write programs* » est explicitement lié au processus de développement et au produit. La deuxième partie « *for people first* » est une contrainte sur l'écriture des programmes visant essentiellement la facilité de maintenance par le personnel dans le futur. La lisibilité des programmes permet au personnel de maintenance de réduire l'effort de compréhension des programmes. La proposition est implicitement un but du management qui pourrait désirer diminuer l'effort de maintenance du logiciel.

Tableau CXCVIII

Éléments d'ingénierie impliqués pour la proposition no.35

Write programs for people first	
Sens explicite	Sens implicite
• Processus • Contrainte • Produit • Ressources (individus)	• But

Proposition no.36: Know software engineering's techniques before using development tools

La proposition vise essentiellement les développeurs à l'effet qu'ils doivent posséder les connaissances du génie logiciel avant de maximiser les avantages des outils de développement. D'une façon implicite, le management est impliqué afin de s'assurer (lors de l'embauche, par exemple) que les candidats possèdent les connaissances requises.

Tableau CXCIX

Éléments d'ingénierie impliqués pour la proposition no.36

Know software engineering's techniques before using development tools	
Sens explicite	Sens implicite
• Ressources	• Contrôle

Proposition no.37: Select tests based on the likelihood that they will find faults

Cette proposition est explicitement liée au processus de test du logiciel. La préparation des jeux d'essais devrait tenir compte de cette proposition. D'une façon implicite, le

contrôle est impliqué, car il fera la vérification (par des mesures) de la pertinence des cas de test.

Tableau CC

Éléments d'ingénierie impliqués pour la proposition no.37

Select tests based on the likelihood that they will find faults	
Sens explicite	Sens implicite
• Processus	• Contrôle • Mesure

Proposition no.38: Choose a programming language to assure maintainability

Le choix du langage, selon Davis (1995), a un impact direct sur l'effort de maintenance du logiciel. La proposition comporte deux volets. Le premier volet concerne la sélection d'un langage de programmation, soit une ressource utilisée par le processus. Le second volet de la proposition souligne la motivation de faire une sélection judicieuse du langage. Ce deuxième volet est à la fois un but du management qui permettrait de diminuer les coûts de la maintenance et une contrainte sur le processus. En effet, certains langages jugés intéressants par les programmeurs pourraient être écartés au profit de d'autres. D'une façon implicite, la proposition a un lien avec le produit final.

Tableau CCI

Éléments d'ingénierie impliqués pour la proposition no.38

Choose a programming language to assure maintainability	
Sens explicite	Sens implicite
• Ressources • Processus • But • Contrainte	• Produit

Proposition no.39: In face of unstructured code, rethink the module and redesign it from scratch

La proposition débute par une constatation (« In face of unstructured code ») qui est le résultat d'une mesure (implicite) faite sur le code d'un programme. Suite à la constatation, la proposition souligne deux actions que le contrôle (implicitement) transmet au processus soit de repenser le module en question et de faire une nouvelle conception au lieu de le modifier. Le produit est aussi impliqué par la proposition.

Tableau CCII

Éléments d'ingénierie impliqués pour la proposition no.39

In face of unstructured code, rethink the module and redesign it from scratch	
Sens explicite	Sens implicite
• Action • Processus	• Contrôle • Produit • Mesure

6.2.4 Synthèse

À la phase 3, les propositions ont été classées selon les catégories processus, produit et individu. Une étape de vérification a été réalisée, dans ce volet, afin de s'assurer de la concordance des résultats avec ce classement. Nous constatons que la catégorie processus regroupe plus d'un élément du modèle de Moore (2006). En effet, cette catégorie regroupe les éléments : processus, contrôle, mesure et action. Le modèle raffine donc encore plus notre catégorie processus. Selon le modèle présenté par Moore (2006), le processus se concentre sur les activités de développement et de maintenance du logiciel alors que le contrôle sur les activités de gestion du processus. Nous remarquons que certaines propositions, classées antérieurement dans la catégorie processus, se retrouvent maintenant associées au contrôle.

Dans le même ordre d'idée, une proposition classée dans la catégorie produit, peut être maintenant associée à l'élément ressource du modèle. Un produit intermédiaire peut être une ressource à un processus subséquent.

Le tableau CCIII présente la synthèse de l'association des propositions avec les éléments du modèle présenté par Moore (2006). Le volet contrôle recueille 21 propositions distinctes, les doublons retranchés du compte. Un doublon est lorsque qu'une proposition est associée aux deux volets. Le volet processus recueille 29 propositions, les doublons également retranchés du compte. Les propositions couvrent majoritairement le volet processus par rapport au volet contrôle. Nous constatons également que 20 propositions couvrent les deux volets. Si nous retranchons ces 20 propositions, le volet contrôle ne conserve que deux propositions et le volet processus que 9. Même en retranchant les propositions communes aux deux volets, le volet processus conserve toujours un plus grand nombre de propositions associées. Cette constatation n'est pas surprenante puisque la majorité des propositions se classe sous la catégorie processus et que ces propositions visent essentiellement le processus de développement du logiciel.

Tableau CCIII

Synthèse de l'association explicite des propositions pour chacun des éléments du modèle de l'ingénierie de Moore (2006)

Volet	Éléments du modèle	Propositions associées explicitement
Contrôle	Contrôle	2, 6, 11, 20, 23, 24, 28, 31, 33
	But	9, 15, 26, 32, 38
	Contrainte	3, 4, 9, 10, 11, 14, 15, 16, 17, 20, 22, 27, 28, 29, 30, 33, 35, 38
	Mesure	2
	Action	16, 39
Processus	Processus	1, 3, 4, 6, 9, 10, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 32, 35, 37, 38, 39
	Ressource	22, 27, 33, 34, 35, 36, 38
	Produit	3, 4, 6, 35
	Besoins	Aucune

Nous constatons que l'élément mesure ne recueille qu'une seule proposition, malgré le fait que la mesure est essentielle dans l'alimentation en information du contrôle afin de détecter des écarts. De plus, l'élément besoin ne recueille aucune proposition.

Au cours du processus d'association des propositions aux éléments du modèle, nous avons aussi identifié, pour chacune des propositions, les éléments qui sont implicitement impliqués dans la proposition. Le tableau CCIV présente les associations implicites des propositions avec le modèle d'ingénierie.

Tableau CCIV

Synthèse de l'association implicite des propositions pour chacun des éléments du modèle de l'ingénierie de Moore (2006)

Volet	Éléments du modèle	Propositions associées implicitement
Contrôle	Contrôle	1, 10, 13, 14, 16, 17, 21, 22, 25, 26, 29, 36, 37, 39
	But	1, 4, 6, 19, 20, 27, 32, 33, 35
	Contrainte	1, 19, 34
	Mesure	1, 6, 13, 16, 20, 23, 28, 31, 37, 39
	Action	1, 2, 6, 20, 21, 23, 28
Processus	Processus	34
	Ressource	1, 4, 9, 21, 24, 26
	Produit	9, 11, 13, 20, 23, 26, 27, 34, 38, 39
	Besoins	2

Nous constatons que la majorité des propositions se retrouve maintenant du côté du volet contrôle. Une seule proposition implique le processus d'une façon implicite. L'élément besoin recueille une proposition.

6.2.5 Conclusion

L'objectif initial de ce premier volet était de vérifier si les 34 propositions retenues de l'analyse couvraient l'ensemble des éléments du modèle d'ingénierie proposé par Moore (2006).

Le volet processus recueille une forte majorité des propositions qui se concentrent essentiellement sur l'élément *processus*. Cette situation peut s'expliquer du fait que la majorité des propositions retenues à la phase 3 se classe dans la catégorie processus. Même si un certain nombre de propositions sont plutôt orientées contrôle, leur nombre demeure faible comparativement aux propositions de type processus. Également, plusieurs propositions ont un sens explicitement orienté développement du logiciel ou soulignant une contrainte sur le processus de développement. Peu de propositions retenues portent sur le contrôle du processus. L'élément *contrôle* ne recueille que neuf propositions à l'opposé de l'élément *processus* qui en comporte 27.

Nous constatons que l'élément *besoin* du volet processus ne recueille aucune proposition. Cette carence est préoccupante compte tenu de l'importance des besoins dans le processus de développement du produit.

Au niveau du volet contrôle, les éléments *mesure* et *action* ne recueillent respectivement qu'une et deux propositions. C'est probablement insuffisant compte tenu que le contrôle est efficace dans la mesure où il est alimenté en données (mesures) afin de pouvoir détecter les écarts et de les corriger sous forme d'action vers le processus.

6.3 Couverture des propositions de principes en fonction du corpus des normes de l'IEEE.

Le deuxième volet de la vérification porte sur le degré de couverture des 34 propositions retenues en fonction du corpus des normes du génie logiciel de l'IEEE.

6.3.1 Objectif

L'objectif de ce volet est d'évaluer le degré de couverture des propositions retenues à l'ensemble du corpus des normes du génie logiciel. Nous pourrons observer qu'elles

sont les normes qui sont bien supportées par les propositions de principes et celles qui le sont moins.

6.3.2 Méthode

Nous basons une partie de notre démarche sur l'ouvrage de James W. Moore (2006) qui propose une synthèse et une organisation du corpus des normes de l'IEEE. Dans un premier temps, nous traitons des 32 propositions de catégorie processus. À titre de rappel, à la phase 3, les propositions ont été classées selon les catégories suivantes : processus, produit et individu. De plus, lors de cette phase, chacune des propositions retenues de la catégorie processus a fait l'objet d'une analyse plus détaillée afin d'identifier à quel processus et activité de la norme ISO/IEC 12207, la proposition est associée. Dans ce volet, nous allons mettre à profit les résultats de cette analyse et l'associer au travail fait par Moore (2006).

Comme le montre la figure 40, la phase 3 de notre analyse a fait des liens entre les propositions de catégorie processus et les processus de la norme ISO/IEC 12207. De son côté, Moore (2006) a fait le travail de prendre chacun des processus de la norme ISO/IEC 12207 et de les associer aux normes du corpus. En combinant, les deux volets des travaux, nous avons une opportunité intéressante de vérifier le degré de couverture de nos propositions sur l'ensemble des normes du génie logiciel de l'IEEE.

faite à la pièce, puisque le chemin de la norme ISO/IEC 12207 n'est pas possible pour celles-ci.

Suite au processus d'association, un tableau synthèse sera proposé présentant l'association entre les propositions et les normes du corpus. Également, un autre tableau exposera, pour chacune des normes, quelles sont les propositions qui s'y rattachent.

Une synthèse des observations sera faite afin de conclure ce 2ième volet de la phase 4.

6.3.3 Corpus des normes du génie logiciel de l'IEEE

Dans son récent ouvrage, James W. Moore (2006) présente le corpus des normes IEEE du génie logiciel qui regroupent 36 normes. Moore présente le corpus selon quatre vues distinctes.

La première vue est l'organisation par sujets des normes. Cette vue a été utilisée par l'IEEE en 1997 pour la publication du corpus. Une contrainte de cette vue est qu'une norme ne peut être classée que dans un seul sujet. Les sujets sont au nombre de huit, tel que présenté au tableau CCV.

Tableau CCV

Sujets pour le classement des normes (Traduit de Moore 2006)

1. Documentation	2. Gestion de projet
3. Processus du cycle de vie	4. Réutilisation
5. Mesures	6. Terminologie
7. Plans	8. Outils

En 1999, l'IEEE propose une organisation différente des normes. Les critères sont le niveau de degré de prescription et les objets. Les niveaux de prescription sont représentés par six catégories présentées à la figure 42.

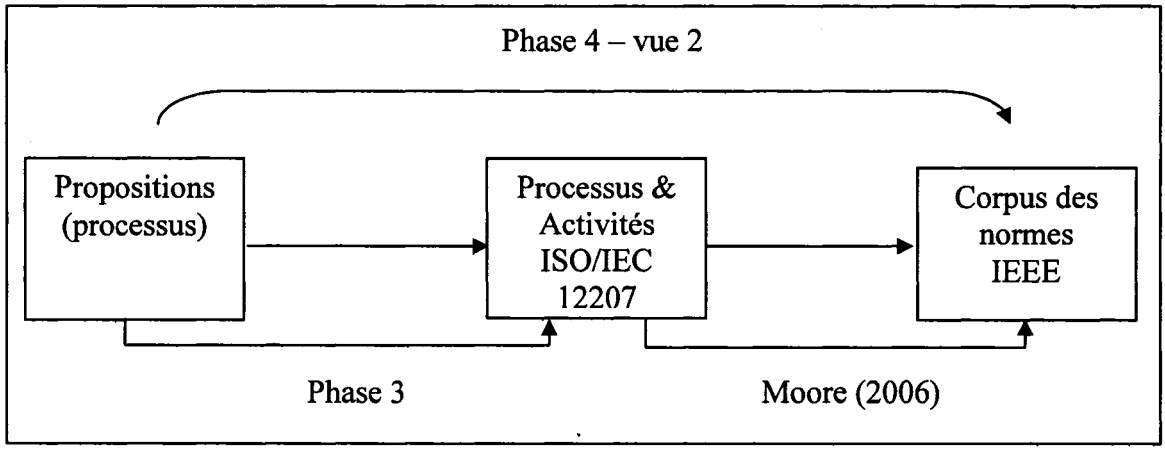

Figure 40 Association des propositions au corpus via la ISO/IEC 12207

La figure 41 présente les éléments de la méthode suivie pour le deuxième volet de la vérification.

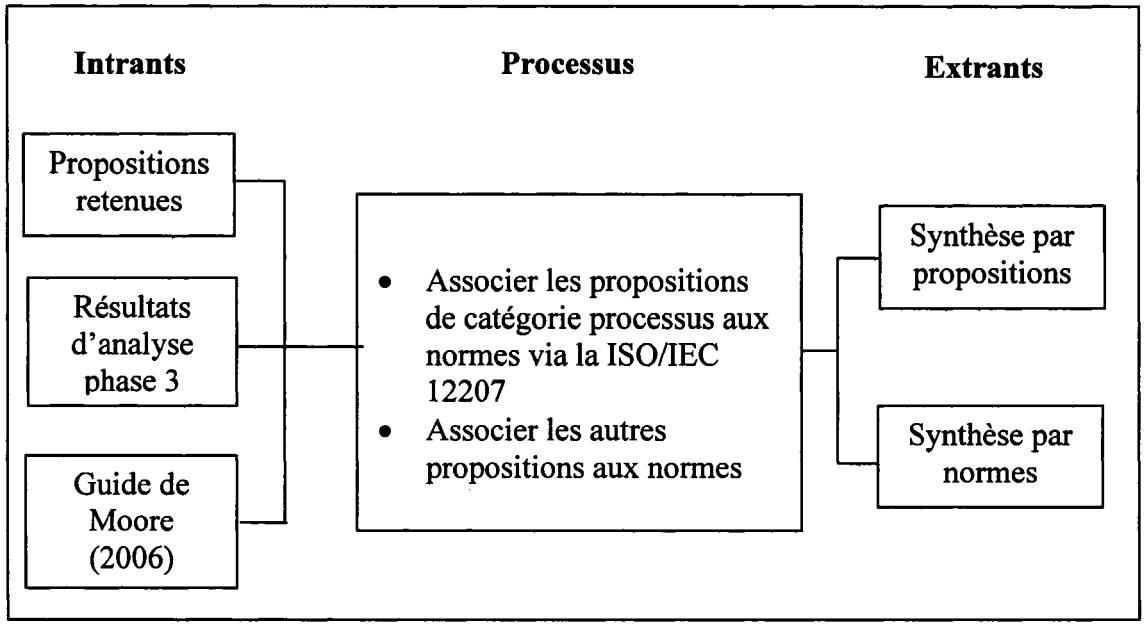

Figure 41 Éléments de la méthode phase 4, volet 2

Du nombre des propositions retenues, seules deux propositions ne sont pas associées à la catégorie processus. Pour ces propositions, la vérification avec le corpus des normes sera

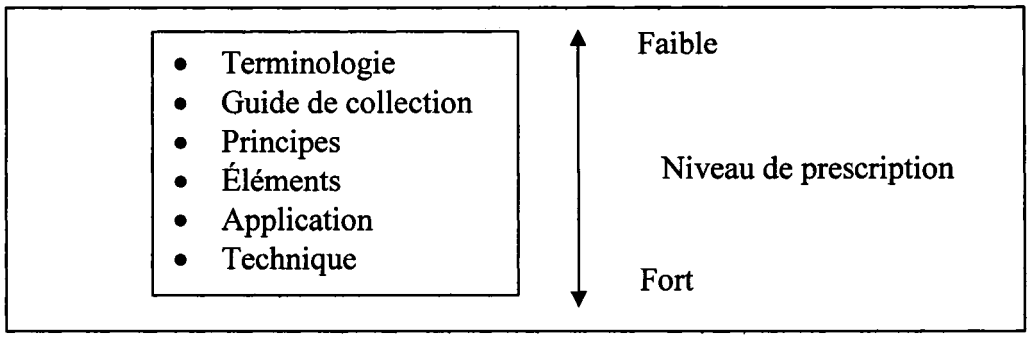

Figure 42 Niveau de prescription (Traduit de Moore 2006)

Les catégories ont un niveau prescriptif allant du plus faible (terminologie) jusqu'au plus fort (technique). Moore (2006) confirme que la grande majorité des normes se situe au niveau de la catégorie élément.

Moore (2006) souligne que les normes peuvent soutenir les quatre principaux objets du génie logiciel interagissant entre eux. Ces objets sont : le client, le processus, les ressources et le produit.

Moore (2006) propose deux nouvelles vues supplémentaires d'organisation du corpus. La troisième vue se fonde sur les dix domaines de connaissances du guide SWEBOK (2004). Chacune des normes est associée à l'un ou l'autre des domaines de connaissances. Une norme peut être associée à plus d'un domaine.

En dernier lieu, Moore propose une quatrième vue du corpus en fonction des processus de la norme parapluie ISO/IEC 12207. Compte tenu que la majorité des propositions que nous avons retenues appartiennent à la catégorie processus, l'organisation par processus du corpus est pertinente pour la vérification du degré de couverture.

Comme nous l'avons déjà présenté antérieurement (à la phase 3), la norme ISO/IEC 12207 structure les processus en trois principales catégories présentées au tableau CCVI.

Tableau CCVI

Catégories de processus ISO/IEC 12207 (Traduction Séguin)

Processus du cycle de vie ISO/IEC 12207		
Primaires	**Soutien**	**Organisationnels**
1. Acquisition 2. Approvisionnement 3. Développement 4. Exploitation 5. Maintenance	1. Documentation 2. Gestion des configurations 3. Assurance qualité 4. Vérification 5. Validation 6. Revue 7. Audit 8. Résolution des problèmes	1. Management 2. Infrastructure 3. Amélioration 4. Formation

6.3.4 Association normes et processus

Nous présentons dans cette section, les résultats des croisements faits par Moore (2006) entre les processus de la norme ISO/IEC 12207 et les autres normes du génie logiciel de l'IEEE. Ces résultats nous sont nécessaires afin de compléter le travail fait par Moore en associant les propositions retenues aux normes du corpus.

6.3.4.1 Processus primaires

Moore propose un raffinement afin de mieux catégoriser les cinq processus primaires. Les processus d'acquisition et d'approvisionnement sont catégorisés de niveau entreprise (affaires) tandis que les processus de développement, d'exploitation et de maintenance sont classés de catégorie technique.

Le tableau CCVII suivant présente la ventilation des normes en fonction en fonction de quatre processus primaire. Le cinquième processus, le développement, fera l'objet d'un tableau distinct.

Tableau CCVII

Association processus primaires et normes (adapté de Moore 2006)

Processus	**Normes associées**
Acquisition	ISO/IEC 12207, IEEE 1517, 1062, 1362
Approvisionnement	ISO/IEC 12207, IEEE 1517
Exploitation	ISO/IEC 12207, IEEE 1517
Maintenance	ISO/IEC 12207, IEEE 1517, 1219, ISO/IEC 14764

Le processus primaire de développement comporte 13 activités. Moore en a fusionné quelques unes puisqu'elles touchaient aux mêmes normes. Le tableau CCVIII suivant présente l'association entre les activités du processus primaire et les normes du corpus.

Tableau CCVIII

Association des activités de développement et normes (adapté de Moore 2006)

Activités du processus de développement	**Normes associées**
Mise en place du processus	ISO/IEC 12207, IEEE1517, 1074
Analyse des exigences système	ISO/IEC 12207, IEEE1517, 1233, 1320.1, 1320.2
Conception architecturale système	ISO/IEC 12207, IEEE1517, 1471
Analyse des exigences du logiciel	ISO/IEC 12207, IEEE1517, 830, ISO9126-1
Conception architecturale du logiciel	ISO/IEC 12207, IEEE1517, 829, 1063, 1471
Conception détaillée du logiciel	ISO/IEC 12207, IEEE1517, 829, 1016, 1063
Programmation et tests du logiciel	ISO/IEC 12207, IEEE1517, 829, 1008, 1063
Intégration du logiciel et tests	ISO/IEC 12207, IEEE1517, 829, 1063
Intégration système et tests	ISO/IEC 12207, IEEE1517, 829
Installation du logiciel et soutien	ISO/IEC 12207, IEEE1517

6.3.4.2 Processus de soutien

La catégorie des processus de soutien comporte huit processus. Moore souligne que les normes ISO/IEC 15939 et l'IEEE 1517 en ajoutent deux autres soit la mesure et la réutilisation. Nous proposons donc d'intégrer les deux processus supplémentaires. Le tableau CCIX suivant présente les résultats de la ventilation des dix processus de soutien en fonction des normes du corpus.

Tableau CCIX

Association des processus de soutien et les normes (Adapté de Moore 2006)

Processus de soutien	**Normes associées**
Documentation	ISO/IEC 12207, IEEE 1063
Gestion des configurations	ISO/IEC 12207, IEEE 828
Assurance qualité	ISO/IEC 12207, 9001, IEEE 730, 1061, 1465 ISO/IEC 90003
Vérification	ISO/IEC 12207, IEEE 1012
Validation	ISO/IEC 12207, IEEE 1012
Revue	ISO/IEC 12207, IEEE 1028
Audit	ISO/IEC 12207, IEEE 1028
Résolution de problème	ISO/IEC 12207, IEEE 1044
Mesure	ISO/IEC 15939, IEEE 982, 1045, 1061, 14143.1
Réutilisation	IEEE 1517, 1420

6.3.4.3 Processus organisationnels

La norme ISO/IEC 12207 définit quatre processus de catégorie organisationnelle. Moore souligne que les normes IEEE 1540 et 1517 en ajoutent chacune un, soit la gestion du risque et la gestion de la réutilisation. Le tableau CCX suivant présente l'association des ces processus avec le corpus des normes de l'IEEE.

Tableau CCX

Association des processus organisationnels et les normes (Adapté de Moore 2006)

Processus	**Normes associées**
Management	ISO/IEC 12207, IEEE 1058, 1490
Infrastructure	ISO/IEC 12207, IEEE 1175, 1462
Amélioration	ISO/IEC 12207, 15504
Formation	ISO/IEC 12207,
Gestion du risque	IEEE 1540
Gestion de la réutilisation	IEEE 1517

6.3.5 Propositions et normes

À la phase 3, les propositions de catégorie processus ont été associées aux processus et aux activités de la norme ISO/IEC 12207. Comme Moore (2006) a fait le lien entre les processus de cette même norme et les autres normes du corpus de l'IEEE, nous allons, dans un premier temps, associer les propositions des catégories processus aux différentes normes du corpus. Dans un deuxième temps, les propositions appartenant uniquement aux normes aux catégories produit et individus seront traitées.

6.3.5.1 Propositions de catégorie processus

Au niveau des phases 2 et 3, nous avons déjà amplement commenté ces propositions. Afin d'éviter la redondance, nous fournissons dans cette section qu'un bref commentaire et le tableau synthèse de l'association entre les processus et les normes pour chacune des propositions. Il est à noter que les normes ISO/IEC 12207 et IEEE 1517 ne seront pas indiquées au niveau des tableaux, compte tenu qu'elles sont associées presqu'à toutes les propositions

Proposition no.1: Align incentives for developer and customer (Davis 1995)

Le sens donné par Davis (1995) à cette proposition est essentiellement orienté sur le développement du code. Le processus primaire de développement est impliqué avec l'activité de programmation et de tests. De plus, la portée étendue englobe aussi le processus de management. Le tableau CCXI présente les résultats de l'association des processus et des normes.

Tableau CCXI

Association processus et normes pour la proposition no.1

Processus	**Activités**	**Normes associées**
Primaire 5.3 Développement	5.3.7 Software coding and testing	IEEE 829, 1008, 1063
Organisationnel 7.1 Management	7.1.2 Planning 7.1.3 Execution and control 7.1.4 Review and evaluation	IEEE 1490, 1058

Proposition no.2: Apply and use quantitative measurements in decision making (Bourque et al. 2002)

La portée de cette proposition touche essentiellement le processus de management et particulièrement deux activités. Il est à noter que la proposition pourrait avoir une portée étendue puisque la prise de décision à l'aide de mesure peut être utilisée dans d'autres processus du cycle de vie.

Tableau CCXII

Association processus et normes pour la proposition no.2

Processus	**Activités**	**Normes associées**
Organisationnel 7.1 Management	7.1.3 Execution and control 7.1.4 Review and evaluation	IEEE 1058, 1490

Proposition no.3 : Build software so that it needs a short user manual (Davis 1995)

La proposition implique le processus primaire de développement et plus spécifiquement quatre de ses activités. Comme le montre le tableau CCXIII, sept normes sont associées à la proposition.

Tableau CCXIII

Association processus et normes pour la proposition no.3

Processus	**Activités**	**Normes associées**
Primaire		
5.3 Développement	5.3.4 Software requirement analysis	IEEE 830, ISO/IEC9126
	5.3.5 Software architectural design	IEEE 829, 1063, 1471
	5.3.6 Software detailed design	IEEE 829, 1016,1063
	5.3.7 Software coding and testing	IEEE 829, 1008, 1063

Proposition no.4 : Build with and for reuse (Bourque et al. 2002)

La proposition est associée à trois activités du processus primaire de développement. À ces activités sont associées cinq normes tel que présenté par le tableau CCXIV.

Tableau CCXIV

Association processus et normes pour la proposition no.4

Processus	**Activités**	**Normes associées**
Primaire		
5.3 Développement	5.3.5 Software architectural design	IEEE 829, 1471, 1063
	5.3.6 Software detailed design	IEEE 829, 1016,1063
	5.3.7 Software coding and testing	IEEE 829, 1008, 1063

Proposition no.6 : Define software artifacts rigorously (Bourque et al. 2002)

La proposition a une portée sur les produits intermédiaires issus du processus primaire de développement. Elle implique quatre activités du processus de développement.

Tableau CCXV

Association processus et normes pour la proposition no.6

Processus	**Activités**	**Normes associées**
Primaire		
5.3 Développement	5.3.4 Software requirement analysis	IEEE 830, ISO/IEC9126
	5.3.5 Software architectural design	IEEE 829, 1471, 1063
	5.3.6 Software detailed design	IEEE 829, 1016,1063
	5.3.7 Software coding and testing	IEEE 829, 1008, 1063

Proposition no.9 : Design for maintenance (Davis 1995)

La proposition implique deux processus primaires soit le développement et la maintenance. Les activités de conception sont ciblées au niveau de ces processus. Même en phase de maintenance, une modification peut nécessiter des changements au niveau de la conception antérieure du logiciel.

Tableau CCXVI

Association processus et normes pour la proposition no.9

Processus	**Activités**	**Normes associées**
Primaire		
5.3 Développement	5.3.5 Software architectural design	IEEE 829, 1471, 1063
	5.3.6 Software detailed design	IEEE 829, 1016, 1063

Tableau CCXVI (Suite)

Processus	Activités	Normes associées
5.5 Maintenance	5.5.2 Problem and modification analysis 5.5.3 Modification implementation 5.5.4 Maintenance review/acceptance	IEEE 1219

Proposition no.10 : Determine requirements now (Davis 1995)

La proposition est associée à deux activités du processus primaire de développement, soit les activités touchant aux exigences. À ces activités, quatre normes sont associées tel que présenté au tableau CCXVII.

Tableau CCXVII

Association processus et normes pour la proposition no.10

Processus	Activités	Normes associées
Primaire		
5.3 Développement	5.3.2 System requirement analysis 5.3.4 Software requirement analysis	IEEE 1233, 1320.1 1320.2 IEEE 830, ISO/IEC9126

Proposition no.13 : Don't try to retrofit quality (Davis 1995)

La proposition est liée à trois processus de soutien : l'assurance qualité, la vérification et la validation.

Tableau CCXVIII

Association processus et normes pour la proposition no.13

Processus	Activités	Normes associées
Soutien		
6.3 Quality Assurance process	6.3.2 Product assurance 6.3.3 Process assurance	IEEE 730, 1061, 1465

Tableau CCXVIII (suite)

Processus	Activités	Normes associées
6.4 Verification	6.4.2 Verification 6.4.2.2 Process verification 6.4.2.3 Requirements verification 6.4.2.4 Design verification 6.4.2.5 Code verification 6.4.2.6 Integration verification 6.4.2.7 Documentation verification	IEEE 1012
6.5 Validation	6.5.2 Validation	IEEE 1012

Proposition no.14 : Don't write your own test plans (Davis 1995)

La proposition est associée aux deux principaux processus primaires, soit le développement et la maintenance.

Tableau CCXIX

Association processus et normes pour la proposition no.14

Processus	Activités	Normes associées
Primaire		
5.3 Développement		
	5.3.7 Software coding and testing	IEEE 829, 1008, 1063
	5.3.9 Software qualification testing	IEEE 829, 1063
	5.3.11 System qualification testing	IEEE 829
5.5 Maintenance		
	5.5.3 Modification implementation	IEEE 1219 ISO/IEC 14764

Proposition no.15 : Establish a software process that provides flexibility (Bourque et al. 2002)

La proposition ne s'applique pas à un groupe de processus en particulier de la norme ISO/IEC12207, mais à l'esprit de la norme dans son ensemble. Cependant, Moore (2006) a identifié un groupe de normes s'appliquant au processus. Ainsi, nous retiendrons donc ces normes pour la proposition tel que présenté au tableau CCXX.

Tableau CCXX

Normes IEEE s'appliquant au processus (Moore 2006)

IEEE 730	IEEE 1045	IEEE 1220
IEEE 828	IEEE 1058	IEEE 12207
IEEE 1008	IEEE 1062	IEEE 1490
IEEE 1012	IEEE 1074	IEEE 1517
IEEE1028	IEEE 1219	IEEE 1540

Proposition no.16 : Fix requirements specification error now (Davis 1995)

La portée de la proposition implique le processus primaire de développement, en particulier, les activités touchant aux exigences. Également, le processus de soutien de vérification est impliqué. Le terme « fix » de la proposition implique qu'une vérification a été faite afin de détecter un écart à combler.

Tableau CCXXI

Association processus et normes pour la proposition no.16

Processus	**Activités**	**Normes associées**
Primaire 5.3 Développement	5.3.2 System requirement analysis 5.3.4 Software requirement analysis	IEEE 1233, 1320.1 1320.2 IEEE 830, ISO/IEC9126
Soutien 6.4 Vérification		IEEE 1012

Proposition no.17 : Give product to customer early (Davis 1995)

La portée de cette proposition, selon Davis (1995), est essentiellement orientée vers le prototypage. Cette technique est utilisée à la phase des exigences afin d'élucider les exigences moins perceptibles et de les valider. Le processus primaire de développement est impliqué, de même que le processus de vérification.

Tableau CCXXII

Association processus et normes pour la proposition no.17

Processus	**Activités**	**Normes associées**
Primaire 5.3 Développement	5.3.4 Software requirement analysis	IEEE 830, ISO/IEC9126
Soutien 6.4 Vérification		IEEE 1012

Proposition no.19: Grow systems incrementally (Davis 1995)

La proposition touche essentiellement le processus primaire de développement. Malgré que Davis (1995) utilise le terme « system » dans la formulation de la proposition, son explication porte essentiellement sur le logiciel.

Tableau CCXXIII

Association processus et normes pour la proposition no.19

Processus	**Activités**	**Normes associées**
Primaire		
5.3 Développement	5.3.4 Software requirement analysis	IEEE 830, ISO/IEC9126
	5.3.5 Software architectural design	IEEE 829, 1471, 1063
	5.3.6 Software detailed design	IEEE 829, 1016, 1063
	5.3.8 Software integration	IEEE 829, 1063
	5.3.9 Software qualification testing	
	5.3.10 System integration	IEEE 829
	5.3.11 System qualification testing	IEEE 829
	5.3.12 Software installation	

Proposition no.20: Implement a disciplined approach and improve it continuously (Bourque et al. 2002)

La proposition n'implique pas un processus en particulier de la norme ISO/IEC 12207, mais l'ensemble des processus. La portée de la proposition est donc de niveau global. Tout comme fait antérieurement pour la proposition no.15, nous utiliserons le regroupement fait par Moore (2006) concernant les normes s'appliquant au processus, tel que présenté au tableau CCXXIV.

Tableau CCXXIV

Normes s'appliquant au processus (Moore 2006)

IEEE 730	IEEE 1045	IEEE 1220
IEEE 828	IEEE 1058	IEEE 12207
IEEE 1008	IEEE 1062	IEEE 1490
IEEE 1012	IEEE 1074	IEEE 1517
IEEE1028	IEEE 1219	IEEE 1540

Proposition no.21: Invest in the understanding of the problem (Bourque et al. 2002)

La proposition implique quatre processus primaires : l'acquisition, l'approvisionnement, le développement et la maintenance. Dans chacun de ces processus, il y a au moins une activité qui s'intéresse à la compréhension du problème à résoudre. La proposition implique le processus de soutien de résolution de problèmes, ainsi que le processus de management de niveau organisationnel.

Tableau CCXXV

Association processus et normes pour la proposition no.21

Processus	Activités	Normes associées
Primaire		
5.1 Acquisition	5.1.1 Initiation	IEEE 1062 IEEE 1362
5.2 Approvisionnement	5.2.1 Initiation	
	5.2.2 Preparation of response	
5.3 Développement	5.3.2 System requirements analysis	IEEE 1233, 1320.1 et .2
	5.3.3 System Architectural design	IEEE 1471 IEEE 830
	5.3.4 Software Requirements analysis	ISO/IEC 9126
5.5 Maintenance	5.5.2 Problem and modification analysis	IEEE 1219 ISO/IEC 14764
Support		
6.8 Problem resolution	6.8.2 Problem resolution	IEEE 1044
Organisationnel		
7.1 Management	7.1.3 Execution and control	IEEE 1490, 1058

Proposition no.22: Involve the customer (Royce 1970)

L'ensemble des processus primaires est impliqué par la proposition. De plus, trois processus de soutien sont impliqués : vérification, validation et revue.

Tableau CCXXVI

Association processus et normes pour la proposition no.22

Processus	Activités	Normes associées
Primaire		
5.1 Acquisition	5.1.1 Initiation 5.1.2 Request for proposal preparation 5.1.3 Contract preparation 5.1.4 Supplier monitoring 5.1.5 Acceptance and completion	IEEE 1062, 1362
5.2 Approvisionnement	5.2.4 Planning	
5.3 Développement	5.3.2 System requirements analysis 5.3.3 System Architectural design 5.3.4 Software Requirements analysis 5.3.5 Software Architectural design 5.3.10 System integration 5.3.11 System Qualification testing 5.3.12 Software installation 5.3.13 Software Acceptance support	IEEE 1233, 1320 IEEE 1471 IEEE 830, ISO/IEC 9126 IEEE 829, 1471, 1063 IEEE 829 IEEE 829
5.4 Exploitation	5.4.3 System operation 5.4.4 User support	IEEE 1517
Support		
6.4 Verification	6.4.2.1 Contract verification 6.4.2.2 Process verification 6.4.2.3 Requirements verification 6.4.2.4 Design verification 6.4.2.6 Integration verification 6.4.2.7 Documentation verification	IEEE 1012
6.5 Validation 6.6 Joint review process	6.5.2 Validation	IEEE 1012 IEEE 1028

Proposition no.23: Keep design under intellectual control (Davis 1995)

Davis (1995) limite grandement la portée de cette proposition à l'utilisation de certaine technique de documentation du design. De ce fait, le sens donné par l'auteur au contrôle intellectuel est très limitatif.

Trois activités sont visées au niveau du processus primaire de développement. Également, la gestion de configuration est aussi impliquée en tant que processus de soutien. Le tableau CCXXVII présente les résultats.

Tableau CCXXVII

Association processus et normes pour la proposition no.23

Processus	**Activités**	**Normes associées**
Primaire 5.3 Développement	5.3.3 System Architectural design 5.3.5 Software Architectural design 5.3.6 Software Detailed design	IEEE 1471 IEEE 829, 1471, 1063 IEEE 829, 1016, 1063
Support 6.2 Configuration management		IEEE 828

Proposition no.24: Maintain clear accountability for results (Boehm 1993)

La proposition implique le processus organisationnel de management et particulièrement les activités de planification. Deux normes sont ainsi associées à la proposition tel que présenté au tableau CCXXVIII.

Tableau CCXXVIII

Association processus et normes pour la proposition no.24

Processus	Activités	Normes associées
Organisationnel 7.1 Management	7.1.2 Planning 7.1.2.1 a, d et e	IEEE 1058, 1490

Proposition no.25: Produce software in a stepwise fashion (Bourque et al. 2001)

La proposition suggère de développer le logiciel en suivant une séquence d'étapes. À ce sujet, la norme ISO/IEC 12207 découpe le processus de développement en plusieurs activités tel que montré au tableau CCXXIX. La portée de la proposition est essentiellement limitée au développement, ainsi, la maintenance n'est incluse.

Tableau CCXXIX

Association processus et normes pour la proposition no.25

Processus	Activités	Normes associées
Primaire 5.3 Développement	5.3.2 System requirements analysis	IEEE 1233, 1320
	5.3.3 System Architectural design	IEEE 1471 IEEE 830, ISO/IEC9126
	5.3.4 Software requirement analysis	IEEE 829, 1471, 1063
	5.3.5 Software architectural design	IEEE 829, 1016,1063
	5.3.6 Software detailed design	IEEE 829, 1008, 1063
	5.3.7 Software coding and testing	IEEE 829, 1063
	5.3.8 Software integration	
	5.3.9 Software qualification testing	IEEE 829
	5.3.10 System integration	IEEE 829
	5.3.11 System qualification testing	
	5.3.12 Software installation	
	5.3.13 Software Acceptance support	

Proposition no.26: Quality is the top priority; long-term productivity is a natural consequence of high quality (Wiegers 1996)

La proposition est liée aux trios groupes de processus de la norme ISO/IEC 12207. Le tableau CCXXX présente les processus et les normes associées.

Tableau CCXXX

Association processus et normes pour la proposition no.26

Processus	**Activités**	**Normes associées**
Primaire		
5.3 Développement	5.3.7 Software coding and testing	IEEE 829, 1008, 1063
	5.3.8 Software integration	IEEE 829, 1063
	5.3.9 Software qualification testing	
	5.3.10 System integration	IEEE 829
	5.3.11 System qualification testing	IEEE 829
5.5 Maintenance	5.5.3 Modification implementation o 5.5.3.2 a) et b)	IEEE 1219 ISO/IEC 14764
Support		
6.3 Quality assurance process		ISO/IEC 9001, 90003 IEEE 730, 1061, 1465
6.4 Verification		IEEE 1012
6.5 Validation process		IEEE 1012
6.6 Joint review		IEEE 1028
6.7 Audit		IEEE 1028
Organisationnel		
7.1 Management	7.1.2.1 g	IEEE 1490, 1058
7.3 Process improvement		ISO/IEC 15504

Proposition no.27 : Rotate people through product assurance (Davis 1995)

La proposition vise essentiellement la gestion des ressources humaines. En effet, elle suggère au gestionnaire de procéder, à l'occasion, à l'envoie d'excellents développeurs faire un séjour au sein du groupe d'assurance qualité. Le tableau CCXXXI présente les processus et les normes associées à la proposition.

Tableau CCXXXI

Association processus et normes pour la proposition no.27

Processus	**Activités**	**Normes associées**
Organisationnel 7.1 Management	7.1.2 Planning o 7.1.2.1 c), d) et e)	IEEE 1490, 1058

Proposition no.28 : Since change is inherent to software, plan for it and manage it (Bourque et al. 2002)

La norme ISO/IEC 12207 prévoit certaines activités pour gérer les changements. Les trois groupes de processus de la norme sont impliqués tel que présenté au tableau CCXXXII.

Tableau CCXXXII

Association processus et normes pour la proposition no.28

Processus	**Activités**	**Normes associées**
Primaire 5.3 Développement	5.3.1 Process implementation • b)	IEEE 1074
5.5 Maintenance		IEEE 1219 ISO/IEC 14764

Tableau CCXXXII (Suite)

Processus	Activités	Normes associées
Support 6.2 Configuration management	6.2.3.1 Change control	IEEE 828
Organisationnel 7.1 Management	7.1.3 Execution and Control	IEEE 1490, 1058

Proposition no.29 : Since tradeoffs are inherent to software engineering, make them explicit and document it. (Bourque et al. 2002)

Les compromis soulignés par la proposition sont principalement effectués au niveau du processus de développement et de la maintenance du logiciel.

Tableau CCXXXIII

Association processus et normes pour la proposition no.29

Processus	Activités	Normes associées
Primaire 5.3 Développement	5.3.2 System requirements analysis 5.3.3 System Architectural design 5.3.4 Software requirement analysis 5.3.5 Software architectural design 5.3.6 Software detailed design 5.3.7 Software coding and testing	IEEE 1233, 1320 IEEE 1471 IEEE 830, ISO/IEC9126 IEEE 829, 1471, 1063 IEEE 829, 1016, 1063 IEEE 829, 1008, 1063
5.5 Maintenance	5.5.2 Problem and modification analysis 5.5.3 Modification implementation	IEEE 1219 ISO/IEC 14764

Proposition no.30: Strive to have a peer, rather than a customer, find a defect (Wiegers 1996)

La proposition vise les activités de développement et de maintenance du logiciel. Également, le processus de soutien avec l'assurance qualité, la vérification et la revue.

Tableau CCXXXIV

Association processus et normes pour la proposition no.30

Processus	**Activités**	**Normes associées**
Primaire		
5.3 Développement	5.3.7 Software coding and testing	IEEE 829, 1008, 1063
	5.3.8 Software integration	IEEE 829, 1063
	5.3.9 Software qualification testing	IEEE 829
	5.3.10 System integration	IEEE 829
	5.3.11 System qualification testing	
5.5 Maintenance	5.5.3 Modification implementation o 5.5.3.2 a) et b)	IEEE 1219 ISO/IEC 14764
Support		
6.3 Quality Assurance	6.3.2 Product assurance	IEEE 730, 1061, 1465, ISO/IEC 9001, 90003
6.4 Verification	6.4.2.3 Requirements verification	IEEE 1012
	6.4.2.4 Design verification	
	6.4.2.5 Code verification	
	6.4.2.6 Integration verification	
	6.4.2.7 Documentation verification	
6.5 Validation		IEEE 1012
6.6 Joint review		IEEE 1028

Proposition no.31: Tailor cost estimation methods (Davis 1995)

L'estimation des coûts et de l'effort est une activité faisant partie de la planification définie par le processus de catégorie organisationnel de management.

Tableau CCXXXV

Association processus et normes pour la proposition no.31

Processus	Activités	Normes associées
Organisationnel 7.1 Management	7.1.2 Planning	IEEE 1490, 1058

Proposition no.32 : To improve design, study previous solutions to similar problems (Bourque et al. 2002)

La proposition est principalement liée au processus primaire de développement, particulièrement à l'activité de conception.

Tableau CCXXXVI

Association processus et normes pour la proposition no.32

Processus	Activités	Normes associées
Primaire 5.3 Développement	5.3.3 System Architectural design 5.3.5 Software architectural design 5.3.6 Software detailed design	IEEE 1471 IEEE 829, 1471, 1063 IEEE 829, 1016, 1063

Proposition no.33 : Use better and fewer people (Boehm 1983)

La proposition implique la gestion des ressources humaines d'un projet. Ainsi, l'activité de planification du processus organisationnel de management est associée à la proposition.

Tableau CCXXXVII

Association processus et normes pour la proposition no.33

Processus	**Activités**	**Normes associées**
Organisationnel 7.1 Management	7.1.2 Planning	IEEE 1490, 1058

Proposition no.34 : Use documentation standards

La proposition implique le processus de soutien de documentation. La norme IEEE/EIA 12207.1 est un supplément à la norme ISO/IEC 12207 afin de préciser la documentation à produire à chacune des étapes du processus. La norme IEEE 1063 se spécialise dans la documentation utilisateur du logiciel.

Tableau CCXXXVIII

Association processus et normes pour la proposition no.34

Processus	**Activités**	**Normes associées**
Soutien 6.1 Documentation		IEEE 1063 ISO/IEC 12207, IEEE/EIA 12207.1

Proposition no.35 : Write programs for people first

La proposition est en lien avec les processus d'écriture du code qui se retrouvent au niveau du développement et de la maintenance tel que présenté au tableau CCXXXIX.

Tableau CCXXXIX

Association processus et normes pour la proposition no.35

Processus	Activités	Normes associées
Primaire		
5.3 Développement	5.3.7 Software coding and testing	IEEE 829, 1008, 1063
5.5 Maintenance	5.5.3 Modification implementation o 5.5.3.2 a) et b)	IEEE 1219 ISO/IEC 14764

Proposition no.37 : Select tests based on the likelihood that they will find faults (Davis 1995)

La proposition implique le processus primaire de développement et de maintenance, particulièrement les activités de tests et d'intégration du logiciel ainsi que l'implémentation des changements dans le contexte de la maintenance.

Tableau CCXL

Association processus et normes pour la proposition no.37

Processus	Activités	Normes associées
Primaire		
5.3 Développement	5.3.7 Software coding and testing 5.3.8 Software integration 5.3.10 System integration	IEEE 829, 1008, 1063 IEEE 829, 1063 IEEE 829
5.5 Maintenance	5.5.3 Modification implementation o 5.5.3.2 a)	IEEE 1219 ISO/IEC 14764

Proposition no.38 : Choose a programming language to assure maintainability

La portée de la proposition vise essentiellement le choix d'outils de développement. Cette activité est liée au processus organisationnel d'infrastructure tel que présenté au tableau CCCLI

Tableau CCXLI

Association processus et normes pour la proposition no.38

Processus	**Activités**	**Normes associées**
Organisationnel 7.2 Infrastructure	Choix des outils et logiciels	IEEE 1175, 1462

Proposition no.39 : In face of unstructured code, rethink the module and redesign it from scratch.

Le sens de la proposition est orienté particulièrement dans un contexte de maintenance, plus particulièrement sur l'implémentation des changements.

Tableau CCXLII

Association processus et normes pour la proposition no.39

Processus	**Activités**	**Normes associées**
Primaire 5.5 Maintenance	5.5.3 Modification implementation o 5.5.3.2 a)	IEEE 1219 ISO/IEC 14764

6.3.5.2 Propositions de catégorie produit et individu

Suite à l'association des 32 propositions de catégorie processus aux corpus des normes, il nous reste maintenant que deux propositions à associer. Pour ce faire, nous utiliserons le classement par objets du génie logiciel proposé par Moore (2006).

Proposition no.11 : Don't overstrain your hardware

À la phase 3, cette proposition a été classée à la catégorie produit. C'est le logiciel qui provoque la surcharge du matériel, donc le produit final. Pour le produit, Moore associe les normes présentées au tableau CCXLIII.

Tableau CCXLIII

Normes associées à la proposition no.11

IEEE 982.1, IEEE 1061, IEEE1063 et IEEE1465

Proposition no. 36 : Know software engineering's techniques before using development tools

La proposition vise les compétences des développeurs à l'effet qu'ils devraient connaître, comprendre et maitriser les techniques de base du génie logiciel. Les normes peuvent fournir un guide pour l'utilisation des outils, mais elles présupposent que l'individu maitrise les techniques sous-jacentes du génie logiciel. De ce fait, nous ne ferons pas d'association de la proposition avec le corpus des normes. La proposition est plutôt orientée vers la formation des individus

6.3.6 Synthèse sur l'association des propositions aux normes

L'objectif de ce second volet de la vérification était d'évaluer le degré de couverture des propositions retenues avec le corpus des normes du génie logiciel de l'IEEE. Les 32 propositions de catégorie processus ont été associées par l'entremise de la norme ISO/IEC 12207. Par la suite, les deux propositions restantes de catégorie produit et individu ont été traitées. Parmi le groupe des 34 propositions, seulement deux n'ont pu être associées (no.31 et no.36) puisque leur signification n'implique pas directement des normes.

Le tableau CCXLIV suivant présente le sommaire des normes associées à chacune des propositions.

Tableau CCXLIV

Synthèse de l'association des propositions aux normes

No.	Proposition	Catégorie
1	Align incentives for developer and customer IEEE 829, 1008, 1058, 1063, 1490,	Individu Processus
2	Apply and use quantitative measurements in decision making IEEE 1058, 1490	Processus
3	Build software so that it needs a short user manual IEEE 829, 830, 1008, 1016, 1063, 1471, ISO/IEC 9126	Produit Processus
4	Build with and for reuse IEEE 829,1008, 1016, 1063, 1471, 1517	Produit Processus
6	Define software artifacts rigorously IEEE 829, 830, 1008, 1016, 1063, 1471, ISO/IEC 9126	Produit Processus
9	Design for maintenance* IEEE 829, 1016, 1063, 1219, 1471	Processus
10	Determine requirements now* IEEE 830, 1233, 1320.1, 1320.2, ISO/IEC9126	Processus
11	Don't overstrain your hardware IEEE 982.1, 1061, 1063, 1465	Produit
13	Don't try to retrofit quality IEEE 730, 1012, 1061, 1465	Processus
14	Don't write your own test plans* IEEE 829,1008, 1063, 1219, ISO/IEC14764	Processus
15	Establish a software process that provides flexibility IEEE 730, 828, 1008, 1012, 1028, 1045, 1058, 1062, 1074, 1219, 1220, 12207, 1490, 1517, 1540	Processus
16	Fix requirements specification error now* IEEE 830, 1012, 1233, 1320.1, 1320.2, ISO/IEC9126	Processus
17	Give product to customers early* IEEE830, 1012, ISO/IEC9126	Processus
19	Grow systems incrementally IEEE 829, 830, 1016,1063, 1471, ISO/IEC9126	Processus
20	Implement a disciplined approach and improve it continuously IEEE 730, 828, 1008, 1012, 1028, 1045, 1058, 1062, 1074, 1219, 1220, 12207, 1490, 1517, 1540	Processus

Tableau CCXLIV (suite)

No.	Proposition	Catégorie
21	Invest in the understanding of the problem IEEE 830, 1044, 1058, 1062, 1219, 1233, 1320.1, 1320.2, 1362, 1471, 1490, ISO/IEC9126, ISO/IEC14764	Processus
22	Involve the customer IEEE 829, 830, 1012, 1028, 1062, 1063, 1233, 1320.1, 1320.2, 1362, 1471, 1517, ISO/IEC9126	Processus
23	Keep design under intellectual control* IEEE 828, 829, 1016,1063, 1471	Processus
24	Maintain clear accountability for results* IEEE 1058,1490,	Processus
25	Produce software in a stepwise fashion* IEEE 829, 830, 1008,1016, 1063, 1233, 1320.1, 1320.2, 1471, ISO/IEC9126	Processus
26	Quality is the top priority; long term productivity is a natural consequence of high quality IEEE 730, 829, 1008,1012, 1028, 1058, 1061, 1063, 1219, 1490, ISO/IEC14764, 15504, ISO 9001, 90003,	Individu Processus
27	Rotate (top performer) people through product assurance IEEE 1058, 1490	Processus
28	Since change is inherent to software, plan for it and manage it IEEE 828, 1058, 1074, 1219, 1490, ISO/IEC14764	Processus
29	Since tradeoffs are inherent to software engineering, make them explicit and document it* IEEE 829, 830, 1008, 1016, 1063, 1219, 1233, 1320.1, 1320.2, 1471, ISO/IEC9126, 14764	Processus
30	Strive to have a peer, rather than a customer, find a defect IEEE 730, 829, 1008, 1028, 1061, 1063, 1219, 1465 ISO/IEC14764, ISO 9001, 90003	Processus
31	Tailor cost estimation methods IEEE 1058 et 1490	Processus
32	To improve design, study previous solutions to similar problems IEEE 829, 1016, 1063, 1471,	Processus
33	Use better and fewer people IEEE 1058, 1490	Individu Processus
34	Use documentation standards IEEE 1063, 12207.1	Produit
35	Write programs for people first IEEE 829, 1008, 1063, 1219, ISO/IEC14764	Produit Processus

Tableau CCXLIV (suite)

No.	Proposition	Catégorie
36	Know software engineering's techniques before using development tools Aucune	Individu
37	Select tests based on the likelihood that they will find faults IEEE 829, 1008, 1063 , 1219, ISO/IEC14764	Processus
38	Choose a programming language to assure maintainability IEEE 1175, 1462	Produit Processus
39	In face of unstructured code, rethink the module and redesign it from scratch.* IEEE1219, ISO/IEC14764	Processus

Le tableau CCXLV présente un sommaire des résultats pour chacune des normes.

Tableau CCXLV

Synthèse de l'association des normes aux propositions

Normes IEEE	Propositions
610.12	Toutes les propositions
730	13, 15, 20, 26
828	15, 20, 23, 28
829	1, 3, 4, 6, 9, 14, 19, 22, 23, 25, 26, 29, 30, 32, 35, 37
830	3, 6, 10, 16, 17, 19, 21, 22, 25, 29
982.1	11
1008	1, 3, 4, 6, 14, 15, 20, 25, 26, 29, 30, 35, 37
1012	13, 15, 16, 17, 20, 22, 26
1016	3, 4, 6, 9, 19, 23, 25, 29, 32
1028	15, 20, 22, 26, 30
1044	13, 21
1045	15, 20
1058	1, 2, 15, 20, 21, 24, 26, 27, 28, 31, 33
1061	11, 13, 26
1062	15, 20, 21, 22
1063	1, 3, 4, 6, 9, 11, 14, 19, 22, 23, 25, 26, 29, 30, 32, 35, 34, 37
1074	15, 20, 28
1175	38
1219	9, 14, 15, 20, 21, 26, 28, 29, 30, 35, 37, 39

Tableau CCXLV (suite)

Normes IEEE	**Propositions**
1220	15, 20
1228	22
12207.1	34
1233	10, 16, 21, 22, 25, 29
1320	10, 13, 21, 22, 25, 29
1362	21, 22
14143.1	
1420	
1462	38
1465	11, 13, 26
1471	4, 6, 9, 19, 21, 22, 23, 25, 29, 32
1490	1, 2, 15, 20, 21, 24, 26, 27, 28, 31, 33
1517	1, 3, 4, 6, 9, 10, 14, 15, 16, 17, 19, 20, 21, 22, 23, 25, 26, 28, 29, 30, 32, 37
1540	15, 20
2001	
Normes ISO	
12207	1, 2, 3, 4, 6, 9, 10, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 37, 38, 39
14764	9, 14, 21, 26, 28, 29, 30, 35, 37, 39
15504	26
9001	13, 26
90003	13, 26
9126	3, 6, 10, 16, 17, 19, 21, 25, 29

Nous constatons que trois normes ne sont pas associées à aucune proposition :

- IEEE 1420
- IEEE 14143.1
- IEEE 2001

Pour la suite, nous utilisons les catégories (objets du génie logiciel) présentés par Moore (2006) afin d'évaluer le degré de couverture des propositions pour chacune des catégories.

La catégorie « client » comporte cinq normes. Chacune de celles-ci est couverte par au moins une proposition. Au total, huit propositions distinctes (i.e. sans doublon) se rattachent aux normes de la catégorie client, soit 24% des propositions. Le tableau CCXLVI présente les résultats.

Tableau CCXLVI

Propositions associées aux normes de catégorie Client

Client		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 1062	4	8
IEEE 1220	2	
IEEE 1228	1	
IEEE 1233	6	
IEEE 1362	2	

La catégorie « produit » comporte six normes qui sont toutes couvertes par au moins une proposition. Pris d'une façon distincte, 21 propositions sont associées aux normes de la catégorie produit, soit 62% des propositions. Le tableau CCXLVII présente les résultats.

Tableau CCXLVII

Propositions associées aux normes de catégorie Produit

Produit		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 982.1	1	21
IEEE 1044	2	
IEEE 1061	15	
IEEE 1063	2	
IEEE 1465	10	
IEEE 9126	2	

La prochaine catégorie, les « ressources », regroupe dix normes. Trois de ces normes ne recueillent aucune proposition, soit : IEEE 1420, IEEE 14143.1 et IEEE 2001. Le

nombre de propositions distinctes se rattachant à la catégorie ressource est de 21, soit 62% des propositions. Le tableau CCXLVIII présente les résultats.

Tableau CCXLVIII

Propositions associées aux normes de catégorie Ressources

Ressources		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 829	15	21
IEEE 830	10	
IEEE 1016	9	
IEEE 1175	1	
IEEE 1320	6	
IEEE 14143	0	
IEEE 1420	0	
IEEE 1462	1	
IEEE 1471	10	
IEEE 2001	0	

La catégorie « processus » regroupent le plus grand nombre de normes avec 16 normes. Chacune de celles-ci est couverte par au moins une proposition. Le nombre de propositions distinctes se rattachant à cette catégorie est de 32, soit 94% des propositions. Compte tenu que 32 propositions sur 34 sont de catégorie processus, nous ne sommes pas surpris de constater un fort degré de couverture auprès des normes de nature processus. Le tableau CCXLIX présente les résultats.

Tableau CCXLIX

Propositions associées aux normes de catégorie Processus

Processus		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 730	4	32
IEEE 828	4	
IEEE 1008	12	
IEEE 1012	7	

Tableau CCXLIX (suite)

Processus		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 1028	5	32
IEEE 1044		
IEEE 1045	2	
IEEE 1058	11	
IEEE 1074	3	
IEEE 1219	11	
IEEE 12207	31	
IEEE 1490	11	
IEEE 1517	22	
IEEE 1540	2	
ISO/IEC 14767	9	
ISO/IEC 90003	2	
ISO/IEC 15504	1	

Le tableau CCL suivant présente les normes qui ont reçues au moins 10 propositions.

Tableau CCL

Normes ayant obtenues 10 propositions et plus

Normes	Nombre de propositions	Catégorie	Normes	Nombre de propositions	Catégorie
IEEE12207	31	processus	IEEE 1058	11	Ressources
IEEE 1517	22	processus	IEEE 1490	11	Ressources
IEEE 1063	15	Produit	IEEE 1471	10	Ressources
IEEE 829	15	Ressource	IEEE 830	10	Ressources
IEEE 1008	12	processus	IEEE 9126	10	Produit
IEEE 1219	11	processus	IEEE 1016	10	Ressources

Nous constatons que trois de quatre catégories de normes sont couvertes par les normes présentées au tableau précédent. Seule la catégorie « client » n'est pas représentée dans ce tableau.

6.4 Conclusion

L'objectif de la phase 4 était d'évaluer le degré de couverture des 34 propositions retenues sous deux angles différents. Le premier angle a permis de vérifier le degré de soutien des propositions par rapport aux éléments de base du modèle d'ingénierie présenté par Moore (2006). Le deuxième angle a permis de vérifier la couverture des propositions avec le corpus de normes du génie logiciel de l'IEEE.

Pour le premier angle de vue, nous avons associé les propositions aux différents éléments du modèle du génie de Moore (2006). Le volet processus du modèle (comprenant aussi le produit et les ressources) recueille la majorité des propositions avec 30, tandis que le volet contrôle en comporte 21. Nous avons constaté que les éléments « besoins » et « mesure » recueillent respectivement zéro et une seule proposition. Ces deux éléments du modèle sont les moins couverts par les propositions. Le tableau CCLI présente la synthèse de l'association des propositions aux éléments du modèle du génie de Moore (2006).

Tableau CCLI

Synthèse de l'association des propositions au modèle d'ingénierie

Volet	Éléments du modèle	Nombre de propositions	Nombre de propositions distinctes
Contrôle	Contrôle	9	21
	But	5	
	Contrainte	18	
	Mesure	1	
	Action	2	
Processus	Processus	28	30
	Ressource	7	
	Produit	4	
	Besoins	0	

Il est à noter que l'élément « contrainte » recueille plus de proposition que l'élément contrôle. Cependant, c'est une caractéristique du génie de réaliser des projets en tenant compte d'un ensemble de contraintes.

Nous constatons que l'ensemble des propositions trouve une correspondance avec les éléments du modèle d'ingénierie. Le fait que la majorité des propositions soit associée à l'élément « processus » du modèle confirme la catégorisation faite à la phase 3 de l'analyse.

Le deuxième angle de notre vérification a consisté à associer les propositions au corpus de normes du génie logiciel de l'IEEE. Pour réaliser ce travail, nous avons choisi de prendre la vision par processus, dans un premier temps, et par objets présentées par Jim Moore (2006). La vision par processus a été principalement utilisée, compte tenu que la majorité des propositions sont de catégorie processus.

À la phase 3 de l'analyse des propositions, nous avons fait le travail d'associer chacune des propositions de catégorie processus aux différents processus de la norme ISO/IEC 12207. Moore (2006) a fait un travail d'associer les processus de cette norme aux autres normes du corpus de l'IEEE. En jumelant nos travaux respectifs, nous avons été en mesure d'établir des liens directs entre les propositions et les normes du corpus et d'en vérifier le degré de couverture.

Nous constatons, tout comme au niveau de la vérification avec le modèle du génie, que les normes appartenant à groupe processus sont les plus supportées par les propositions. Cette constatation reconfirme de nouveau la justesse de la catégorisation faite à la phase 3. Le tableau CCLII suivant présente une synthèse du degré de couverture selon les quatre regroupements (par objets) présentés par Moore (2006).

Tableau CCLII

Synthèse du degré de couverture par regroupements de l'IEEE

Groupes de normes IEEE	Degré de couverture des propositions
Processus	94%
Produit	62%
Ressources	62%
Client	24%

Les normes des groupes « Produit » et « Ressources » sont aussi bien couverte par les propositions avec un taux de 62% pour chacun des groupes. Le groupe « Client » ne recueille qu'un taux de couverture de 24%. Cependant, ce plus faible niveau de couverture s'explique par le fait que la majorité des propositions est de catégorie processus.

CHAPITRE 7

SYNTHÈSE DES RÉSULTATS

7.1 Introduction

Dans ce dernier chapitre, nous dressons un bilan de cette recherche analytique sur les principes fondamentaux du génie logiciel. En premier lieu, nous revenons brièvement sur l'état de la question avant le déroulement de cette thèse. De ce constat, les objectifs de cette recherche qui en découlent sont résumés, de même que la méthodologie choisie sera commentée suite à sa mise en application pour chacune de ses étapes. Par la suite, nous présentons une synthèse des résultats obtenus et la suite des travaux possibles pour poursuivre la recherche sur les principes fondamentaux du génie logiciel.

7.2 Constatation de la situation avant le déroulement de la recherche

En 2002, Bourque et al. (2002) présentaient les résultats d'une recherche empirique sur les principes fondamentaux du génie logiciel. Cette recherche fut commandée par la Computer Society, plus particulièrement le groupe des normes (Moore 1998, 2006). Cependant, la liste des principes candidats proposés s'ajoutait aux nombreuses autres listes de principes proposés par d'autres auteurs depuis 1970.

Suite à la revue de littérature, nous avons recensé 308 principes sur une période allant de 1970 jusqu'à 2003. Nous n'avons pas la prétention d'avoir tout recensé, mais la grande majorité l'a été. Il est très peu probable qu'une discipline du génie, tel le génie logiciel, comporte autant de principes, dits fondamentaux. De plus, à la lecture et à l'analyse des travaux antérieurs, nous avons constaté un certain nombre de lacunes importantes que nous résumons ici.

En premier lieu, la majorité des études ne définit pas le terme principe. Cette lacune a entraîné des conséquences importantes sur les résultats proposés par les auteurs. D'une part, cette absence de définition a conduit à une confusion dans l'utilisation des termes principe, concept, lois et technique. Pour plusieurs auteurs, se sont des termes synonymes, alors que ce n'est pas le cas, comme nous l'avons présenté à la section 3.3 du chapitre 3. D'autre part, l'absence de définition du terme principe s'est aussi fait ressentir dans la formulation des principes proposés qui se présente sous trois formes. La première formulation est un seul mot représentant plutôt un concept. La deuxième forme est une proposition descriptive et la troisième forme est une proposition prescriptive guidant l'action. Nous avons constaté que deux auteurs (Bourque et al. 2002 et Davis 1995) sur les quatre qui ont donné une définition au terme principe ont présenté des propositions et non des concepts. Parmi les autres auteurs, les trois formulations sont présentes, parfois même pour le même auteur.

En deuxième lieu, nous avons constaté une faiblesse méthodologique dans plusieurs travaux. Ainsi, en plus de l'absence de définition, peu d'auteurs ont présenté leur méthode pour identifier les principes et soutenir leurs résultats. Seuls les travaux de Bourque et al. (2002) présentent explicitement la méthode suivie pour arriver aux résultats proposés. De plus, seulement deux auteurs ont proposé explicitement des critères d'identification des principes. L'absence de définition, de critères d'identification et de méthodologie explicite fait en sorte que nous pouvons nous questionner sur l'origine des principes proposés. Dans certains cas, nous n'avons aucune idée sur la façon dont les auteurs sont arrivés à ces résultats. Ainsi, nous avons fait l'hypothèse que plusieurs listes de principes proviendraient tout simplement de l'opinion personnelle des auteurs.

En dernier lieu, nous avons constaté que certains auteurs ne font pas une nette distinction entre les concepts du génie logiciel avec ceux de l'informatique. Nous pensions au début que plus les travaux remontaient dans le temps, plus cette situation serait fréquente

compte tenu de la jeunesse du génie logiciel. Cependant, nous avons constaté que même des études récentes telles Meyer (2001) comportent cette situation. De plus, les auteurs ne partagent pas nécessairement la même définition et la même vision du génie logiciel. Ainsi, nous avons constaté que certains principes étaient plutôt liés au domaine de l'informatique que du génie logiciel.

Dans cet état de la situation, la recherche sur les principes fondamentaux du génie logiciel ne pouvait avancer qu'avec difficulté. D'une part, il y a trop de principes proposés pour la discipline. Même les travaux structurés et documentés de Bourque et al. (2002) sur la question n'ajoutent qu'une autre liste de principes. Ainsi, si un groupe de chercheurs désirent creuser la question des principes, la question se posera d'elle même : quelle liste de principes doit-on choisir? De plus, en choisir une en particulier au détriment des autres listes, peut amener la critique de la part de la communauté. Ainsi, cette constatation était la motivation fondamentale de cette thèse.

Il était donc difficile de poursuivre les travaux sur les principes sans traiter de front la quantité imposante de principes recensés par l'entremise d'une méthodologie de recherche analytique rigoureuse afin d'en arriver à une liste réduite respectant des critères précis. Ainsi, était le but de cette thèse.

7.3 Retour sur les objectifs

Dans le but de faire progresser la recherche sur les principes fondamentaux, nous avions comme objectif de concevoir une méthodologie de recherche rigoureuse composée de phases afin d'évaluer d'une façon systématique les nombreuses listes de principes proposées par les auteurs depuis 1970. Cette méthodologie de recherche a permis d'éliminer les faux principes et de ne conserver que ceux qui satisfont à la définition choisie du terme principe et aux critères d'identification. Il est à noter que n'avions pas l'objectif d'identifier de nouveaux principes.

Nous avons aussi réalisé les objectifs qui découlent des lacunes constatées dans les travaux antérieurs. Les objectifs réalisés sont :

- Définir les termes principe et concept
- Définir les critères d'identification
- Définir le génie logiciel et identifier ses concepts fondamentaux
- Identifier les concepts de l'informatique
- Vérifier si les principes retenus soutiennent les éléments de base du génie et les normes du génie logiciel de l'IEEE

7.4 La méthodologie choisie pour la recherche

Afin de réaliser les objectifs fixés, nous avons conçu une méthodologie de recherche composée de quatre étapes principales qui ont été présentées au chapitre 2. Le tableau CCLI présente en résumé ces étapes.

Tableau CCLIII

Étapes principales de la méthodologie

Phases	**Description**
1) Définition du cadre conceptuel d'analyse	• Concepts du génie • Définition du génie logiciel et identification des concepts • Identification des activités du génie logiciel • Définitions : concept, principe, lois • Critères d'identification des principes
2) Évaluation selon les critères individuels	• Application des critères individuels d'identification des principes

Tableau CCLIII (suite)

Phases	Description
3) Évaluation selon les critères d'ensemble et liens avec la norme ISO/IEC12207	• Catégorisation des principes • Application des critères d'ensemble d'identification des principes • Liens avec la norme ISO/IEC 12207
4) Évaluation du degré de couverture des principes retenus	• Avec les éléments d'un modèle de l'ingénierie • Avec les normes du génie logiciel de l'IEEE

Nous allons maintenant présenter un bilan de la réalisation des phases de la méthodologie de recherche sous forme de bilan

7.4.1 Phase 1 - Définition du cadre conceptuel

Le résultat de recherche de cette phase est la définition du cadre conceptuel et la description de ses éléments. Le cadre conceptuel conçu est composé de six éléments principaux tel que présentés à la figure 43.

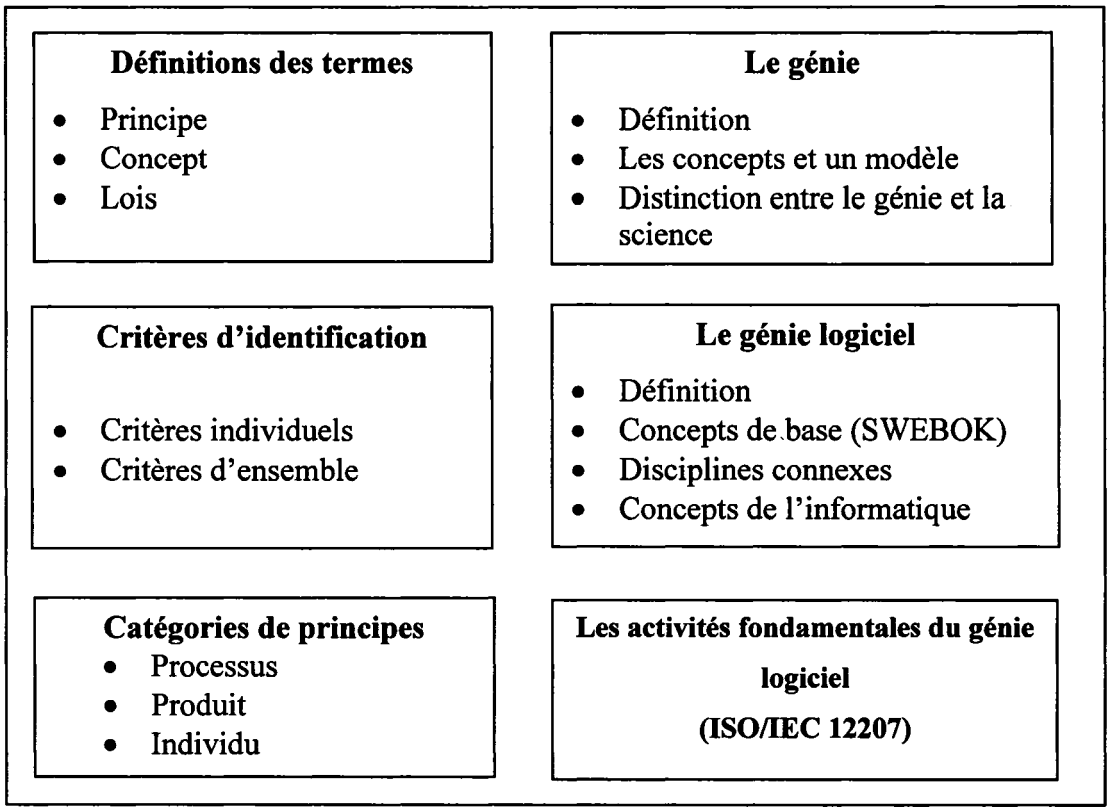

Figure 43 Éléments du cadre conceptuel

En premier lieu, les concepts généraux du génie ont été identifiés. Les principes du génie logiciel peuvent découler de principes du génie. Ainsi, certains concepts du génie peuvent être présents dans la formulation des principes. Nous avons donc jugé important d'identifier à la section 3.1 du chapitre 3, les concepts généraux du génie afin de les détecter lors de l'analyse des principes proposés.

Le deuxième élément du cadre défini est le génie logiciel, à la section 3.2, du chapitre 3. Comme nous avons constaté que les auteurs recensés pour cette étude ne partagent pas tous la même définition et la même vision du génie logiciel, il était incontournable de prendre position sur ce sujet. D'une part, nous avons choisi la définition de l'IEEE pour

L'identification des concepts a permis de les détecter dans la formulation des principes. Un des critères d'identification des principes stipule qu'un principe du génie logiciel doit comporter dans sa formulation un ou plusieurs concepts de la discipline. Ainsi, l'identification des concepts du génie logiciel a permis d'appliquer ce critère d'identification. Les concepts ont été présentés sous forme de tableaux à la section 3.2.1 du chapitre 3. À titre de rappel, comme tous les principes analysés sont formulés en anglais, les concepts du génie logiciel ont été laissés volontairement en anglais pour éviter toute imprécision de traduction et d'interprétation.

Parmi les disciplines frontières du génie logiciel identifié par SWEBOK, l'informatique est la discipline frontière auquel nous avons apporté une attention particulière. Nous avons donc procéder à l'identification des concepts de cette discipline afin d'être en mesure de les identifier dans la formulation des principes étudiés.

Le troisième élément du cadre, traité à la section 3.2.4 du chapitre 3, a été d'identifier les activités fondamentales du génie logiciel. Certains principes recensés sont en fait que des activités normées à être réalisées. Dans ces cas, il ne s'agit plus de principe, mais de simples activités à réaliser. Nous avons choisi la norme parapluie ISO/IEC 12207, norme adoptée également par l'IEEE. Cette norme identifie et décrit les principaux processus et activités à réaliser dans un projet de génie logiciel.

Le quatrième élément du cadre, les définitions des termes principe et concept, a été un peu plus complexe à définir. Afin de pallier au manque de définition des termes constatés dans plusieurs travaux antérieurs, ce quatrième élément du cadre est une pièce fondamentale pour soutenir l'analyse des principes. Nous avons recensé plus de vingt définitions du terme principes et treize pour le terme concept. Nous avons donc été en mesure de composer une définition pour ces termes et de les distinguer entre eux à la section 3.3 du chapitre 3. Ainsi, la définition retenue du terme principe est la suivante :

Proposition fondamentale de la discipline formulée sous forme prescriptive (règle), à la source des actions, pouvant être vérifiée dans ses conséquences et par l'expérience.

Le cinquième élément du cadre est la définition des critères d'identification permettant de retenir un principe ou de l'écarter. Nous avons identifié cinq critères individuels d'identification s'appliquant à chacun des principes pris individuellement. De plus, deux critères d'ensemble ont été définis. Le tableau CCLII présente ces critères.

Tableau CCLIV

Liste des critères retenus

Critères d'identification individuels

1. Un principe est une proposition formulée de façon prescriptive
    - Indique quoi faire, mais sans spécifier comment le faire
2. Un principe ne doit pas être associé directement ou découler d'une technologie, d'une méthode ou d'une technique ou être une activité du génie logiciel. (adapté de Bourque et al.(2002))
3. Le principe ne doit pas énoncer un compromis (ou un dosage) entre deux actions ou concepts (Bourque et al.(2002) et de Moore (1998))
4. Un principe du génie logiciel devrait inclure des concepts reliés à la discipline ou au génie. (Bourque et al.(2002))
5. La formulation d'un principe doit permettre de le tester en pratique ou de le vérifier dans ses conséquences. (Bourque et al.(2002))
    - Vérifiable dans ses conséquences et par l'expérience

Critères d'ensemble

1. Les principes devraient être indépendants (non déduit) (Boehm (1983))
2. Un principe ne doit pas contredire un autre principe connu. (Bourque et al. (2002))

Le sixième et dernier élément du cadre porte sur les catégories de principe. Dans la synthèse des travaux antérieurs à cette thèse, nous avions identifié implicitement dans les principes proposés trois catégories : processus, produit et individu. Ces catégories nous ont permis de faire des regroupements principalement à la phase 3 afin de mieux appliquer le critère d'ensemble no. 1 portant sur la déduction d'un principe à partir d'un autre.

Ces six éléments de base du cadre conceptuel sont les pièces maîtresses de l'évaluation en trois phases des principes.

7.4.2 Phase 2 - Évaluation selon les critères individuels

La phase 2 de la méthodologie de recherche a consisté à analyser un à un les 308 principes recensés et les confronter aux principaux éléments du cadre conceptuel, en particulier à la définition retenue et aux cinq critères individuels d'identification. La phase 2 s'est déroulée itérativement, chacune des itérations raffinant un peu plus les résultats. La figure 44 présente un résumé du processus suivi pour cette phase.

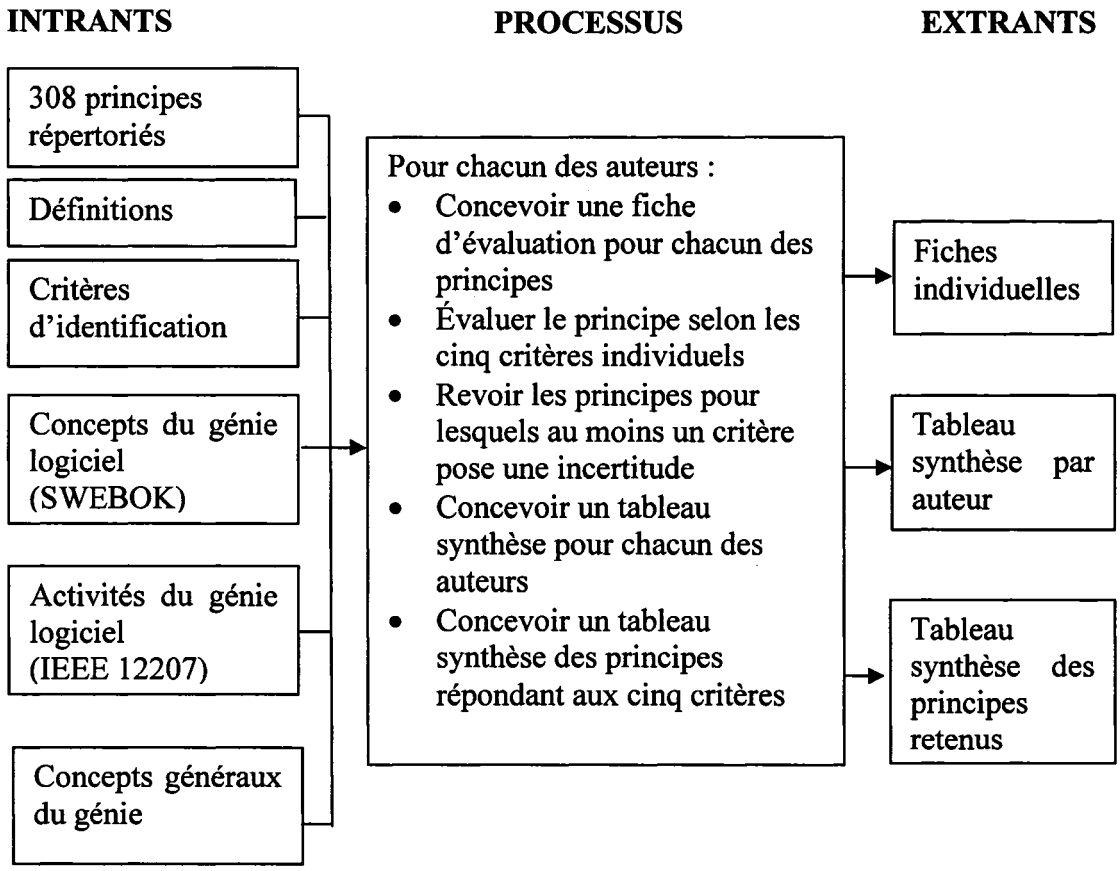

Figure 44 Méthode de recherche de la phase 2

Dans une première itération, l'application des cinq critères individuels a permis d'écarter 273 propositions. Celles-ci ne satisfaisaient pas au moins un des cinq critères individuels. Nous avons noté que quatre des cinq critères avaient permis d'éliminer un nombre relativement équivalent de principes. Le critère no. 3 n'a éliminé que trois propositions. Ainsi, peu de propositions comportaient une forme de dosage ou de compromis entre deux actions à faire.

Nous avons aussi écarté 52 principes qui étaient formulés à l'aide d'un seul mot. Ces principes n'étaient en fait que des concepts. Ainsi, ces concepts sont en contradiction directe avec la définition du terme principe puisqu'ils ne représentent aucunement une proposition prescriptive.

Le critère d'identification no. 2 a permis d'éliminer 133 propositions. La majorité de ces propositions étaient en fait des activités du génie logiciel répertoriées au sein de la norme ISO/IEC 12207. Un principe n'est pas en soi une activité, cependant, une ou des activités peuvent découler d'un principe. Également, certains principes écartés représentaient plutôt des techniques ou des aspects liés à une méthode.

Nous avons écarté 126 propositions qui ne comportaient pas explicitement de concept du génie logiciel (critère no. 4). Lors d'une itération subséquente, nous en avons récupéré quelques unes qui contenaient des concepts généraux du génie.

L'application du critère no. 5 n'a pas été facile. Dans la majorité des cas, la formulation du principe n'inclut pas les conséquences de l'appliquer ou non. Ainsi, il a été nécessaire de consulter l'explication donnée par l'auteur, lorsque disponible, afin d'évaluer les conséquences possibles. Si l'explication ne permettait pas d'identifier les conséquences, nous avons donc évalué si une expérimentation théorique pouvait être faite. Certaines formulations ne permettent pas d'identifier clairement des conséquences. À titre d'exemple, le principe « Case tool are expensive » ne représente pas une formulation vérifiable dans ses conséquences sur le logiciel. Seulement douze principes ont été écartés sur ce critère alors qu'ils satisfaisaient aux quatre autres critères. Ainsi, dans ces cas, la variable dépendante (les conséquences) ne peut être identifiée précisément.

À la fin de la phase 2, nous avons constaté que 18 propositions avaient été écartées sur le critère no. 1, mais celles-ci satisfaisaient les quatre autres critères. Cette constatation nous a amené à porter une attention particulière à ces principes rejetés. Avions-nous

écarté des propositions intéressantes sur la seule base que la formulation n'était pas prescriptive? Nous avons donc fait une itération supplémentaire pour ce groupe de principes afin de déterminer si une reformulation mineure permettrait de les récupérer. Suite à cette itération, quatre propositions sur les 18 ont été récupérées à l'aide d'une reformulation mineure les rendant prescriptives.

Le tableau CCLV présente une synthèse par auteurs des principes proposés et retenus suite à la réalisation de la phase 2.

Tableau CCLV

Sommaire des principes proposés et retenus par auteur

Auteurs	**Principes proposés**	**Principes retenus**	**Lois empiriques**
Winston W. Royce (1970)	5	1	0
Ross, Goodenough et Irvine (1975)	7	0	0
H.D. Mills (1980)	3	0	0
Many Lehman (1980)	5	0	2
Barry W. Boehm (1983)	7	2	0
Booch et Bryan (1984)	7	0	0
Buschmann et al. (1996)	11	0	0
Alan Davis (1995)	201	24	11
Karl Wiegers (1996)	14	2	0
Anthony Wasserman (1996)	8	0	0
Paul Taylor (2001)	10	0	0
Bertrand Meyer (2001)	13	0	0
Bourque, Dupuis, Abran, Moore, Tripp et Wolf (2002)	15	10	0
Ghezzi, Jazayeri et Mandrioli (2003)	7	0	0

Nous avions fait mention d'une hypothèse à l'effet que certaines propositions pourraient être plutôt des lois empiriques que des principes. À titre de rappel, une loi empirique est une généralisation venant essentiellement d'observations de la pratique. Une loi serait moins générale qu'un principe et elle peut ne pas toujours être vraie, contrairement à un principe qui est une vérité fondamentale.

Le tableau CCLVI présente 13 énoncés de lois empiriques potentielles que nous avons identifiées au cours de l'analyse des principes. Nous constatons que le thème du changement se retrouve dans six des propositions. D'autre part, nous constatons que les propositions sont directement liées à des observations et une généralisation faites de la pratique.

Tableau CCLVI

Lois empiriques candidates du génie logiciel

1. The law of continuing change
2. The law of increasing complexity
3. The more seen, the more needed
4. Change during development is inevitable
5. You can reuse without a big investment
6. Half the errors found in 15 percent of modules
7. A few good people are better than many less skilled people
8. People and time are not interchangeable
9. Software will continue to change
10. Software's entropy increases
11. The older a program, the more difficult it is to maintain
12. Language affect maintainability
13. Maintenance causes more errors than development

Il n'était pas dans nos objectifs d'identifier des lois potentielles du génie logiciel. Cependant, des travaux futurs pourraient creuser la question à l'effet d'identifier formellement les lois empiriques du génie logiciel. Les lois empiriques candidates identifiées pourraient servir de départ à ces travaux.

La deuxième phase a donc permis de retenir 39 propositions parmi les 308 présentes au départ. Près de 90% des propositions recensées dans les travaux antérieurs ont été écartées suite à l'application des cinq critères individuels d'identification. Au niveau des annexes, le lecteur retrouvera les fiches individuelles d'analyse pour chacun des 308 principes analysés. Ces tableaux fournissent la traçabilité souhaitée pour aider les chercheurs à mieux comprendre les résultats obtenus.

7.4.3 Phase 3 - évaluation selon les critères d'ensemble et liens avec la norme ISO/IEC12207

La troisième phase avait pour objectif principal d'appliquer les deux critères d'identification portant sur l'ensemble des 39 principes retenus issus de la phase 2. Afin de faciliter l'application de ces critères, les propositions ont été, au préalable, classées selon les catégories identifiées : processus, produit et individu. La figure 45 résume la méthode de recherche suivie pour la phase 3.

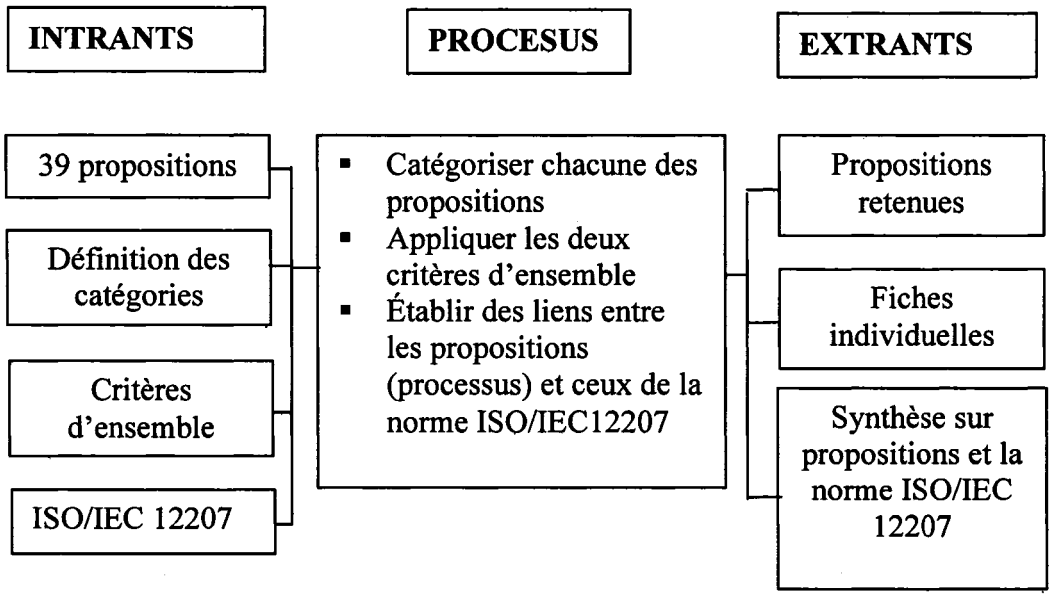

Figure 45 Méthode de recherche de la phase 3

En premier lieu, Les propositions ont été classées selon les trois catégories identifiées. Le tableau CCLVII présente le résumé de cette classification. Une proposition peut se retrouver dans plus d'une catégorie. Nous avons constaté que la grande majorité des propositions retenues (36/39) se classent dans la catégorie processus.

Tableau CCLVII

Ventilation des propositions selon les trois catégories

Catégories	**Nombre**
Processus	31
Produit	6
Individu	4

Par la suite, les deux critères d'identification portant sur l'ensemble des propositions ont été appliqués systématiquement. Le tableau CCLVIII rappelle ces deux critères.

Tableau CCLVIII

Critères d'identification portant sur l'ensemble des propositions

Critères d'ensemble
1. Les principes devraient être indépendants (non déduit) (Boehm)
2. Un principe ne doit pas contredire un autre principe connu. (Bourque et al.)

L'application des critères d'ensemble s'est aussi fait d'une façon itérative. Nous avons fait plusieurs itérations en considérant les résultats et les difficultés rencontrées.

En ce qui concerne le premier critère, suite aux itérations faites, nous avons choisi de modérer quelque peu son application dans l'analyse. Un certain nombre de propositions ont été jugées déduites de propositions plus générales. Cependant, nous avons observé que certaines propositions pouvant être déduites de d'autres comportaient une précision intéressante au sujet de l'action à entreprendre. Une précision que la proposition parent

n'offrait pas. Suite à cette observation, nous avons donc refait une itération en considérant les propositions déduites. Les propositions déduites qui offraient une précision ont été conservées et celles qui n'ajoutaient rien de plus que la proposition parent ont été écartées. Ainsi, dix propositions déduites ont été conservées.

Concernant le deuxième critère, aucune proposition sur les 39 analysées, ne contredit une autre proposition. Ce critère n'a pas donc pas été en mesure d'écarter aucune proposition.

Suite à la réalisation de la phase trois, nous avons conservé 34 propositions, dont dix pouvant être déduites de propositions plus générales, mais ajoutant une précision intéressante. Le tableau CCLIX présente les propositions retenues suite à la réalisation de phase trois.

Tableau CCLIX

Propositions retenues suite à la phase 3 de l'analyse

No.	Proposition
1	Align incentives for developer and customer
2	Apply and use quantitative measurements in decision making
3	Build software so that it needs a short user manual
4	Build with and for reuse
6	Define software artifacts rigorously
9	Design for maintenance*
10	Determine requirements now*
11	Don't overstrain your hardware
13	Don't try to retrofit quality
14	Don't write your own test plans*
15	Establish a software process that provides flexibility
16	Fix requirements specification error now*
17	Give product to customers early*
19	Grow systems incrementally
20	Implement a disciplined approach and improve it continuously
21	Invest in the understanding of the problem

Tableau CCLIX (suite)

No.	Proposition
22	Involve the customer
23	Keep design under intellectual control*
24	Maintain clear accountability for results*
25	Produce software in a stepwise fashion*
26	Quality is the top priority; long term productivity is a natural consequence of high quality
27	Rotate (high performer) people through product assurance
28	Since change is inherent to software, plan for it and manage it
29	Since tradeoffs are inherent to software engineering, make them explicit and document it*
30	Strive to have a peer, rather than a customer, find a defect
31	Tailor cost estimation methods
32	To improve design, study previous solutions to similar problems
33	Use better and fewer people
34	Use documentation standards
35	Write programs for people first
36	Know software engineering's techniques before using development tools
37	Select tests based on the likelihood that they will find faults
38	Choose a programming language to assure maintainability
39	In face of unstructured code, rethink the module and redesign it from scratch.*

* : propositions déduites, mais conservées

Un aspect important qui a été réalisé à la phase trois est le lien fait entre les propositions de catégorie processus et les processus identifiés à la norme ISO/IEC 12207. Pour chacune des propositions de cette catégorie, nous avons identifié les processus associés à la norme. Cette activité nous a amené à évaluer le degré de soutien des propositions retenues face aux processus de la norme ISO/IEC 12207.

Le tableau CCLVIII présente une synthèse de la répartition des propositions de catégories processus selon les processus de la norme ISO/IEC 12207.

Tableau CCLX

Dénombrement des propositions par groupes de processus

Groupes de processus ISO/IEC12207	Nombre de propositions
Primaire (34)	
Développement	20
Maintenance	10
Acquisition	2
Approvisionnement	2
Exploitation	0
Soutien (17)	
Vérification	4
Validation	4
Revue	3
Gestion configuration	2
Assurance qualité	2
Audit	1
Résolution de problème	1
Documentation	0
Organisationnel (11)	
Management	9
Amélioration processus	1
Infrastructure	1
Formation	0

Le tableau montre que les processus primaires de développement et de maintenance sont les processus les mieux soutenus par les propositions retenues. Cependant, nous avons constaté que le processus d'exploitation ne recueille aucune proposition. En effet, il y une forte tendance des propositions retenues sur le processus de création ou de développement du logiciel et non de son exploitation. Le groupe de processus de soutien est bien soutenu, même si le nombre de propositions est moins élevé. Il est à noter que le processus de documentation n'a pas recueilli de proposition. En dernier lieu, le groupe de processus organisationnel regroupe dix propositions. Le processus de formation ne recueille aucune proposition tandis que le processus de management recueille la majorité

des propositions de ce groupe. Nous observons qu'il y a un processus dans chacun des groupes qui ne recueille aucune proposition.

7.4.4 Phase 4 - Évaluation du degré de couverture des principes retenus

Cette phase a été essentiellement une vérification de la liste des propositions retenues à la phase trois. L'objectif de cette phase était de procéder à une vérification de couverture en deux volets.

7.4.4.1 Vérification avec les éléments du modèle d'ingénierie

Le premier volet a consisté à associer les 34 propositions aux éléments de base du modèle du génie présenté par Moore (2006). La question de recherche posée fut : est-ce les propositions retenues soutiennent chacun des éléments du modèle ?

Chacune des 34 propositions a fait l'objet d'une nouvelle analyse afin d'extraire les relations des propositions avec les éléments du modèle. Nous avons donc analysé les liens explicites provenant de la formulation de la proposition. Le tableau CCLIX présente la synthèse de l'association des propositions aux éléments du modèle du génie.

Tableau CCLXI

Synthèse de l'association des propositions pour chacun des éléments du modèle de l'ingénierie

Volet	Éléments du modèle	Propositions associées explicitement
Contrôle	Contrôle	2, 6, 11, 20, 23, 24, 28, 31, 33
	But	9, 15, 26, 32, 38
	Contrainte	3, 4, 9, 10, 11, 14, 15, 16, 17, 20, 22, 27, 28, 29, 30, 33, 35, 38
	Mesure	2
	Action	16, 39

Tableau CCLXI (suite)

Volet	Éléments du modèle	Propositions associées explicitement
Processus	Processus	1, 3, 4, 6, 9, 10, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 32, 35, 37, 38, 39
	Ressource	22, 27, 33, 34, 35, 36, 38
	Produit	3, 4, 6, 35
	Besoins	aucune

Le volet contrôle recueille 21 propositions distinctes (doublons retirés), tandis que le volet processus en recueille 29. Cette situation confirme la catégorisation des propositions faites au niveau de la phase 2 de l'analyse. L'élément mesure ne recueille qu'une seule proposition malgré le fait que la mesure alimente le contrôle en information afin de détecter les écarts. L'élément besoin ne recueille aucune proposition.

7.4.5 Vérification avec les normes de l'IEEE

Pour le deuxième volet de la vérification du degré de soutien des propositions, nous avons l'exercice avec les normes du génie logiciel de l'IEEE. Pour atteindre l'objectif, nous avons utilisé les liens faits à la phase 2 entre les propositions et les processus de la norme ISO/IEC 12207. Par la suite, nous avons utilisé les liens faits par Moore (2006) entre les processus de la norme ISO/IEC 12207 et les autres normes du génie logiciel de l'IEEE. La figure 46 présente la voie retenue pour effectuer les liens entre les propositions et les normes.

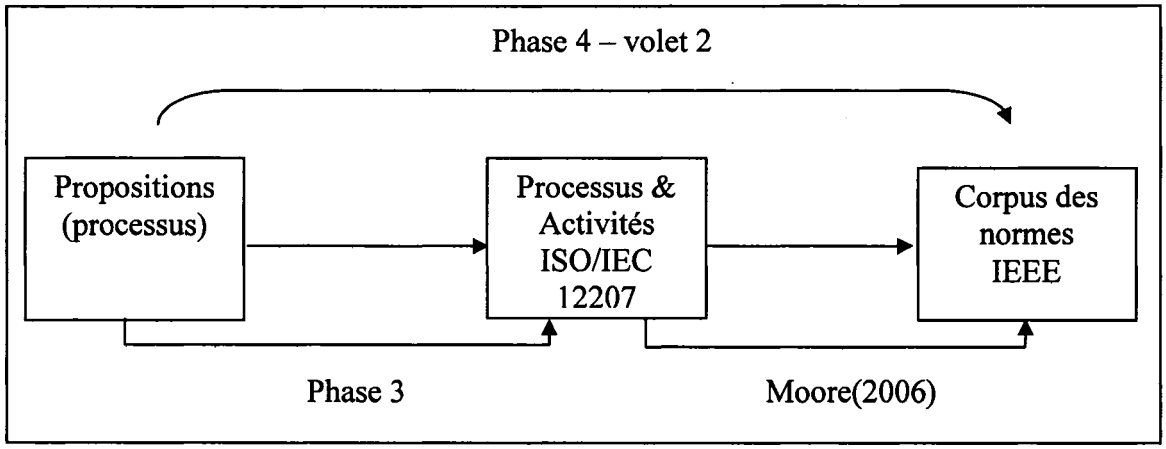

Figure 46 Association des propositions au corpus via la ISO/IEC 12207

La section 6.3 du chapitre 6, présente tous les liens effectués entre les propositions et les normes de l'IEEE et quelques normes ISO. Pour cette synthèse, nous présentons les résultats en fonction des catégories présentées par Moore (2006). Ces catégories, au nombre de quatre, sont : le client, le produit, les ressources et les processus. Les quatre prochains tableaux présentent le nombre de propositions associées à chacune des normes pour les quatre catégories.

Tableau CCLXII

Propositions associées aux normes de catégorie Client

Client		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 1062	4	8
IEEE 1220	2	
IEEE 1228	1	
IEEE 1233	6	
IEEE 1362	2	

Tableau CCLXIII

Propositions associées aux normes de catégorie Produit

Produit		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 982.1	1	21
IEEE 1044	2	
IEEE 1061	15	
IEEE 1063	2	
IEEE 1465	10	
IEEE 9126	2	

Tableau CCLXIV

Propositions associées aux normes de catégorie Ressources

Ressources		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 829	15	21
IEEE 830	11	
IEEE 1016	10	
IEEE 1175	1	
IEEE 1320	6	
IEEE 14143	0	
IEEE 1420	0	
IEEE 1462	1	
IEEE 1471	11	
IEEE 2001	0	

Tableau CCLXV

Propositions associées aux normes de catégorie Processus

Processus		
Normes	Nombre de propositions associées	Nombre de propositions distinctes
IEEE 730	4	32
IEEE 828	4	
IEEE 1008	12	
IEEE 1012	7	
IEEE 1028	5	
IEEE 1044		
IEEE 1045	2	
IEEE 1058	10	
IEEE 1074	3	
IEEE 1219	11	
IEEE 12207	31	
IEEE 1490	10	
IEEE 1517	22	
IEEE 1540	2	
ISO/IEC 14767	9	
ISO/IEC 90003	2	
ISO/IEC 15504	1	

Les normes de la catégorie processus recueillent le plus grand nombre de propositions avec 32. Toutes les normes de cette catégorie sont couvertes par au moins une proposition. La catégorie client est celle qui recueille le moins de soutien de la part des propositions retenues. Cependant, cette constatation ne nous étonne pas compte tenu que la majorité des propositions retenues sont de catégorie processus. De ce fait les résultats observés au niveau des normes ne fait que confirmer le classement fait à la phase 3 de l'analyse des propositions. Le tableau CCL présente les pourcentages de degré de couverture selon les catégories de normes de Moore (2006).

Tableau CCLXVI

Synthèse du degré de couverture par regroupements de l'IEEE

Groupes de normes IEEE	Degré de couverture des propositions
Processus	94%
Produit	62%
Ressources	62%
Client	24%

7.5 Commentaires sur les résultats

La réalisation de cette thèse analytique a permis de traiter, avec une méthodologie de recherche bien définie, les 308 propositions de principe recensées dans les travaux antérieurs. Aucune autre étude connue n'a eu cette envergure afin d'analyser le grand nombre de principes suggérés. Cette analyse en trois phases a permis d'identifier 34 propositions qui répondent aux sept critères d'identification. En retirant les dix propositions pouvant être déduites, il nous que 24 propositions sur les 308 du départ.

Au niveau des points forts de cette recherche, nous soulignons, en premier lieu, la rigueur et la transparence de la méthodologie de recherche utilisée. Les six éléments de base du cadre conceptuel combinés à l'évaluation des principes en trois phases est une méthode riche, étapiste et transparente pour traiter les 308 propositions. Avec cette méthodologie, nous sommes convaincus d'avoir pallié à plusieurs lacunes majeures constatées au niveau de plusieurs travaux antérieurs. Nous considérons que la conception de la méthodologie utilisée pour cette thèse constitue un apport original et important dans la recherche sur les principes du génie logiciel.

Tous les principes retenus sont des propositions prescriptives guidant l'action dans la réalisation de projet de génie logiciel. Ces propositions contiennent un ou des concepts du génie logiciel identifiés par SWEBOK (2004) ou des concepts généraux du génie.

Les propositions retenues ont été ensuite liées, dans un premier temps, avec les éléments de base du modèle de l'ingénierie présenté par Moore (2006). Nous y avons constaté un fort degré de support. Dans un deuxième temps, nous avons fait l'exercice avec les normes du génie logiciel de l'IEEE où le support est excellent.

Nous avons atteint les objectifs fixés au départ de cette recherche. Les objectifs réalisés sont :

- Définir les termes principe et concept;
- Définir les critères d'identification des principes
- Définir le génie logiciel et identifier ses concepts fondamentaux ;
- Identifier les concepts de l'informatique;
- Concevoir une méthodologie de recherche composée de phases pour évaluation rigoureusement et avec transparence les principes répertoriés;
- Vérifier si les principes retenus soutiennent les éléments de base du génie et les normes du génie logiciel de l'IEEE.

Il est maintenant possible de poursuivre les recherches sur les principes du génie logiciel en prenant la liste épurée des 34 propositions retenues. Les travaux futurs n'auront plus à être confrontés aux nombreuses listes de principes. De plus, puisqu'il y maintenant moins de principes à tenir compte, des travaux d'approfondissement sur chacun d'eux sont maintenant possibles et d'une ampleur moindre puisque la liste originale a fondu de près de 90%.

Advenant le cas où une liste de principes publiée antérieurement n'aurait pas été prise en compte dans cette thèse, cette liste pourra être traitée facilement à l'aide de la méthode en trois phases élaborée dans cette thèse. Ainsi, les résultats de cette liste seront vite connus. La méthode n'est pas sujette à une désuétude rapide et peut donc être utilisée dans le futur pour traiter d'autres listes de principes, s'il y a lieu.

Également, nous avons identifié au passage des lois candidates du génie logiciel qui répondaient à la définition du terme loi. À titre de rappel, une loi empirique est une généralisation venant essentiellement d'observations de la pratique. Une loi serait moins générale qu'un principe et elle peut ne pas toujours être vraie, contrairement à un principe qui est une vérité fondamentale. Ainsi, après les principes, le génie logiciel pourrait aussi comporter des lois.

Cette thèse n'a pas la prétention de fournir l'ultime liste des principes fondamentaux du génie logiciel. Il est possible que certains principes du génie logiciel n'aient pas encore été identifiés ou qu'ils ne fussent pas dans les listes de principes recensés dans les travaux antérieurs. Cette thèse n'avait pas pour objectif d'identifier de nouveaux principes.

Maintenant que nous proposons une liste réduite de principes candidats du génie logiciel, la recherche doit maintenant se poursuivre. D'une part, il faut développer un consensus autour de la liste réduite proposée. Cette thèse n'avait pas cet objectif. L'atteinte d'un consensus est une étape importante dans la suite des travaux. Cependant, c'est une étape qui prend un certain temps à se réaliser. D'autre part, les principes candidats proposés devraient être testés et vérifier au niveau de la pratique. Nous souhaitons que ces activités se réalisent dans le futur.

Comme l'ont démontré les résultats, la majorité des propositions appartient à la catégorie processus. Cependant, nous en retrouvons beaucoup moins au niveau des catégories produit et individu. Nous pensons qu'il y a probablement des vides à combler dans ces deux dernières catégories. Nous remarquons que depuis 35 ans, il y a eu beaucoup d'accent de mis sur le processus de réalisation du logiciel afin de produire le logiciel dans les délais prévus et les budgets. Cet accent sur le processus peut être une des raisons de la quantité moindre de principes dans les autres catégories. Nous faisons

donc l'hypothèse que des principes seraient encore à être identifiés dans les catégories produit et individu.

Depuis le démarrage de cette thèse, nous sommes confrontés aux difficultés que pose le traitement de ce type de sujet. Cette thèse n'est pas le développement d'un logiciel ou d'une nouvelle technologie, mais la manipulation et le traitement des concepts à la base de la discipline du génie logiciel. Parmi les difficultés, l'interprétation a été une difficulté constante. Plusieurs auteurs ont identifié des principes, mais ne les ont pas commentés ou expliqués. Dans ces cas, il a été nécessaire de choisir l'interprétation que nous jugions la plus pertinente. De plus, même lorsque les auteurs commentaient les principes proposés, nous avons constaté des imprécisions importantes, voir même des contradictions entre la formulation du principe et son explication. Davis (1995) fait partie de ce groupe pour certains principes qu'il a proposé. De plus, nous avons constaté que parfois l'explication donnée limitait grandement la portée du principe formulé. Dans ces cas, le principe aurait dû être reformulé pour tenir compte des limitations fournies par l'auteur. À la phase 2, nous avons fait l'exercice de reformuler certains principes pour rendre leur formulation prescriptive. Dans ces cas, la reformulation est beaucoup plus près de l'explication fournie par l'auteur que la formulation originale.

7.6 Suite des travaux

Maintenant que nous avons une liste réduite de principes candidats, il est possible de poursuivre et d'approfondir la recherche sur les principes.

Au cours de la dernière année, déjà d'autres travaux ont été entrepris en utilisant comme source de départ la liste des principes issus de cette thèse. En premier lieu, une autre thèse de doctorat (Abran et al. 2006) est en cours portant sur les principes et les domaines de connaissances de l'ingénierie proposés par Vincenti (1990). Cette recherche se base sur la liste des principes issue de cette thèse afin de voir comment ces

principes peuvent être opérationnalisés dans une perspective d'ingénierie. De plus, un raffinement des critères est envisagé.

En deuxième lieu, nous nous sommes intéressés à appliquer la liste des principes candidats au contexte du logiciel libre (open source). Dans un récent article (Séguin et al. 2006), nous avons identifié les principes qui s'appliquent au contexte particulier du logiciel libre et ceux qui ne s'appliquent pas. Certains principes du génie logiciel peuvent être utilisés dans le paradigme du logiciel libre.

Au niveau des travaux futurs, il serait intéressant de développer chacun des principes retenus en les commentant plus en profondeur et en délimitant leur portée. Cet aspect serait important à être réalisé avant de développer éventuellement le consensus autour de cette liste. En fait, les commentaires et la portée permettraient à la communauté de mieux comprendre les propositions. De plus, les explications permettraient aussi de mieux guider l'utilisation des principes dans la pratique et de les intégrer dans la formation.

De plus, il y a un travail à faire au niveau de la formulation des principes. Il faut s'assurer que la portée de la formulation soit identique à la vision de l'auteur. A titre de rappel, nous avons constaté des écarts entre la formulation et l'explication de certains principes proposés par Davis (1995). La majorité des formulations n'expriment pas les conséquences d'appliquer ou non les principes.

Cette thèse n'a pas tenu compte des principes à la base d'une méthode ou d'une idéologie de développement tel Agile ou Xtreme programming. Il serait intéressant d'évaluer ces principes à l'aide de la méthodologie développée pour cette recherche et d'en analyser les résultats.

7.7 Impact industriel

L'objectif de cette thèse n'était pas de développer un nouvel outil de développement, une nouvelle technologie ou une nouvelle méthode, mais d'approfondir la recherche à la base même de la jeune discipline du génie logiciel.

En 1996, la Computer Society avait commandé une recherche sur l'identification des principes du génie logiciel (Moore 1997, 2006). En 2002, Bourque et al. (2002) ont publié les résultats de cette étude. La liste de principes proposés a été intégrée dans cette thèse. Nous sommes maintenant en mesure de proposer nos résultats à la Computer Society.

À cet effet, James W. Moore est un membre actif et influant de la Computer Society. Il est membre du conseil des gouverneurs, il est le co-éditeur exécutif du guide SWEBOK (SWEBOK 2004), il est membre du comité des normes du génie logiciel de l'IEEE et responsable des liaisons entre ce comité et le comité ISO/IEC JTC 1/SC7. En plus d'être Vice-président de « Electronic Products and Services » de la Computer Society, James W. Moore est le président du comité « Professional Practice Commitee ». Actuellement, ce comité explore la définition des pratiques fondamentales du génie logiciel. Ainsi les résultats de cette thèse arrivent à point pour alimenter les travaux du comité. L'architecture de travail mis en place par le comité est présentée à la figure 47. Il est à noter que les résultats de cette thèse servent à définir la boite « Principles of Practice ».

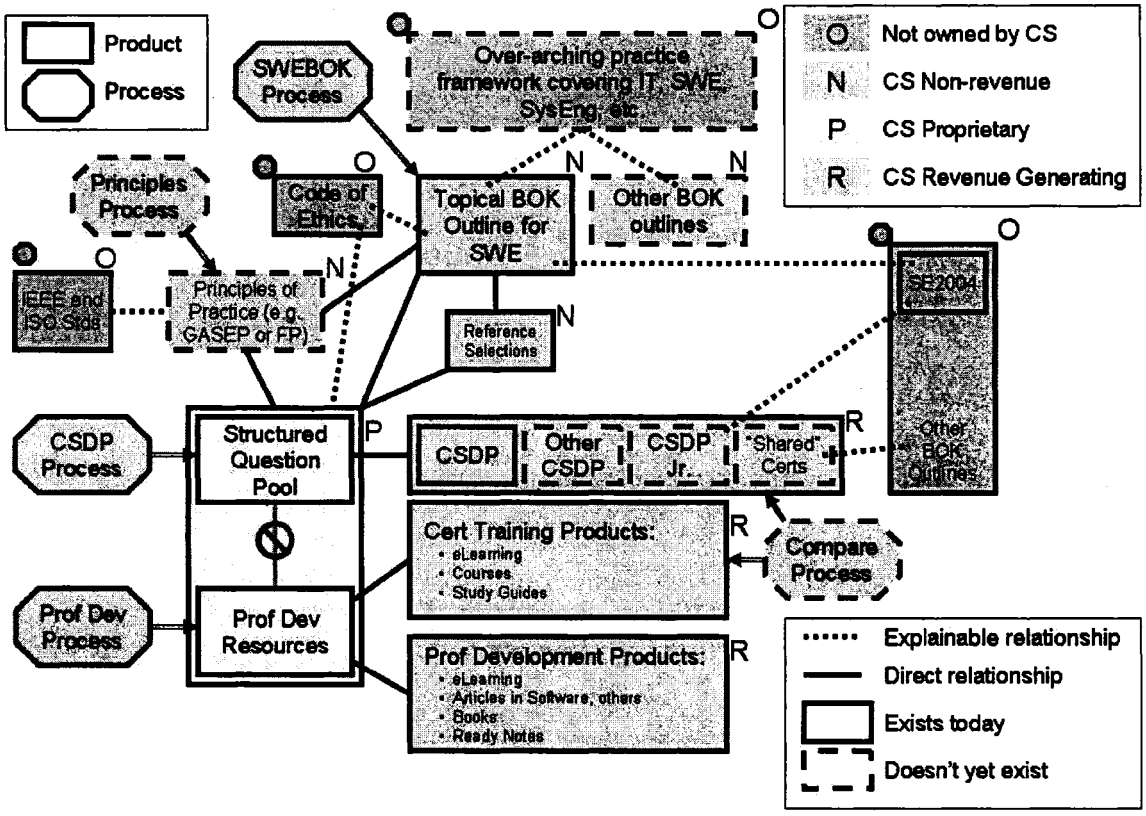

Figure 47 Architecture des travaux du Professional Practices Commitee (IEEE)

Actuellement, les 34 principes retenus de cette thèse ont été intégrés comme base aux travaux du comité. Le comité demandé à ce que je participe, comme expert, aux travaux de développement du contenu de la boite « Principles of practice ».

Nous considérons aussi que la liste des principes retenus peut aussi être utilisée en éducation pour former les nouveaux ingénieurs du logiciel. Nous suggérons qu'un travail soit entrepris pour intégrer les principes dans les différents cours du curriculum du génie logiciel.

CONCLUSION

Cette thèse conclut cinq années de recherche sur les principes fondamentaux du génie logiciel. Nous sommes confiants de voir les résultats obtenus être utilisés par la communauté du génie logiciel. À court terme, le comité des pratiques professionnelles de la Computer Society, piloté par James W. Moore, a déjà intégré la liste des principes obtenus par cette thèse. Dès l'automne 2006, d'autres aspects seront utilisés par ce groupe.

Nous estimons avoir atteint le but de cette thèse et ainsi aider la discipline du génie logiciel à mieux cerner la question des principes fondamentaux du génie logiciel et de contribuer à l'atteinte de sa maturité comme discipline du génie.

ANNEXE 1

Définitions recensées

Les fichiers de l'annexe 1 sont sur le CD dans le répertoire annexe-1

ANNEXE 2

Fiches individuelles – évaluation phase 2

Les fichiers de l'annexe 2 sont sur le CD dans le répertoire annexe-2.

ANNEXE 3

Fiches individuelles – reformulation phase 2

Les fichiers de l'annexe 3 sont sur le CD dans le répertoire annexe-3.

ANNEXE 4

Fiches individuelles – évaluation phase 3

Les fichiers de l'annexe 4 sont sur le CD dans le répertoire annexe-4.

ANNEXE 5

Fiches de travail – évaluation phase 4

Les fichiers de l'annexe 5 sont sur le CD dans le répertoire annexe-5.

BIBLIOGRAPHIE

Abran, A., Moore, J.W., Bourque, P., Dupuis, R., Tripp,L.L. (ed. 2004). *Software Engineering Body of Knowledge*. Los Alamitos : IEEE Computer Society Press.

Abran, A., Séguin, N., Bourque, P., Dupuis, R. (2004) *The Search for Software Engineering Principles: An Overview of Results*. in Conference on Principles of Software Engineering - PRISE 2004 , Buenos Aires, Argentine , 51-60 .

Abran, A., Meridji, K. (2006). Analysis of Software Engineering from An Engineering Perspective. *European Journal for the Informatics Professional* ,vol. 7, No. 1, February, 46-52 .

Anane, R. (2000). Software Egineering and History. *Conference SAC2000 ACM*, March, 19-21, 137-140.

Aslaksen, E.W. (1996). *The changing nature of engineering*. Sydney: McGraw-Hill

Boehm, B.W. (1983). Seven Basic Principles of Software Engineering. *The Journal of Systems and Software*, Vol.3, no 1, May, 366-371.

Booch, G, Bryan, D. (1994). *Software Engineering with Ada*. (3^{e} éd.). California : Benjamin/Cummings Publishing.

Bourque, P.; Dupuis, R.; Abran, A.; Moore, J.W.; Tripp, L.L.; Wolff, S. (2002). Fundamental Principles of Software Engineering - A Journey. *Journal of Systems and Software*, vol. 62, No. 1, 59-70.

Brooks, F.P. (1995). *Le mythe du mois-homme : essais sur le génie logiciel*. Paris : Thompson Publishing.

Bryant, A. (2000). Methaphor, myth and mimicry : The bases of software engineering. *Annals of Software Enginnering*, Vol 10, 273-292.

Bryant, A. (2000). It's Engineering Jim... but not as we know it. *Proceedings of the 22nd international conference on Software engineering*. ACM. 78-87.

Bunge, M. (2003). *Philosophical Dictionary*. New York : Prometheus Books.

Bushcmann, F., Meunier, R, Rohnert, H., Sommerlad, P., Stal, M. (1996). *Pattern-Oriented Software Architecture*. England : Wily.

Coda, F., Ghezzi, C., Vigna, G., Garzotto, F. « Towards a software Engineering Approach to Web Site Development. *9th International Workshop on Software Specification & Design*, IEEE, 1-9.

Cowling, A.J. (2001). Structuring the Disciplines Related to Software Engineering : A general Model. *14th Conference on Software Engineering Education and Training*. IEEE Computer Society, 231-239.

Davis, A.M. (1994). Fifteen Principles of Software Engineering. *IEEE Software*, Novembre, 94-101.

Davis, A.M. (1995). *201 Principles of Software Developement*. New-York : MgGraw-Hill.

Davis, M. (1998). *Thinking Like an Engineer*. New-York : Oxford University Press.

Denning, P.J. (2001). Who Are We?. *Communcations of the ACM*, February, Vol 44, No 2, 15-19.

Diaz-Herrera, J.L. (2001). Engineering Design for Software : On Defining the Software Engineering Profession. *31st ASEE/IEEE Frontiers in Education Conference*. IEEE Computer Society.

Dubois, J. (2002). *Lexis - Larousse de la langue française*. Paris : Larousse.

Duggins, S., Thomas, B.B. (2002). An Historical Investigation of Graduate Software Engineering Curriculum. *Proceedings of the 15th Conference on Software Engineering Education and Trainig (CSEET'02)*. IEEE Computer Society.

Dupuis, R.; Bourque, P.; Abran, A.; Moore, J.W. (1997). Principes fondamentaux du génie logiciel: Une étude Delphi in Le génie logiciel et ses applications. *Dixièmes journées internationales (GL97)*, Paris, 3-5, décembre.

Encyclopaedia Unversalis (2002). Encyclopaedia Unversalis. Vol 5, Paris : Universalis.

Ghezzi, C., Jazayeri, M., Mandrioli, D. (2003). *Fundamentals of Software Engineering*. (2^{e} éd.). New-Jersey : Prentice Hall.

Gilb, T. (1988). Principles of Software Engineering Management. Don Mills : Addison-Wesley.

Glass, R.L. (2003). Facts and Fallacies of Software Engineering. New York :Addison-Wesley.

Grimson, J.B., Kugler, H.J. (2000). Software Needs Engineering – a position paper. *Proceedings of the 22nd international conference on Software engineering.* ACM. 541-544.

Hamlet, D, Maybee, J. (2001). *The engineering of Software.* New York : Addison-Wesley

Hempel, C.G. (1966). *Philosophy of Natural Science.* Englewood Cliffs : Prentice-Hall Inc.

Hilburn, T.B. (1996). Software Engineering – from the Beginning. 9th Conference on Software Engineering Education (CSEE), IEEE Computer Society, 29-39.

Hilburn, T.B., Bagert, D.J. (1999). A Software Engineering Curriculum Model. *29e ASEE/IEEE Frontiers in ducation Conference.* IEEE.

Hoffman, D.M., Weiss, D.M. (2001). *Software Fundamentals – Collected Papers by David L. Parnas.* New York : Addison-Wesley

IEEE 610.12: 1991. (1991). *IEEE Standard Glossary for Software Engineering Terminology.* USA : IEEE Computer Society.

ISO/IEC 12207:1995. (1995). *International Standard 12207 – Information Technology: Software Life Cycle Processes.* Genève : International Organization for Standardization – ISO.

Jabir; Moore, J.W.; Abran, A.; Bourque, P.; Group, Business Planning; Dupuis, R.; Hybertson, D.; Jacquet, J.-P.; Köller, A.; Lowry, E.; Tripp, L.L. (1998), A Search for Fundamental Principles of Software Engineering - Report of a Workshop conducted at the Forum on Software Engineering Standards Issues, Montréal, Québec, Canada, 21-25 Octobre 1996. *Computer Standards and Interfaces*, vol. 19, no. 2, Mars, 155-160.

Johnston, L. (2001). Learning from traditional architects. *The proceedings of INTERACT2001*, 1-7

Kuhn, T.S. (1970). *The Structure of Scientific Revolutions.* 2ième edition, Vol 2, Numéro 2, Chicago : The University of Chicago Press.

Lehman, M. (1980). On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle. *Journal of Systems and Software*, July, Vol1, No 3, 213-221.

Lewerentz, C., Rust, H. (2000), Are software engineers true engineers? *Annals of Software Enginnering,* Vol 10, 311-328.

Littré. E. (2004). *Le Littré : dictionnaire de la langue française en un volume.* Paris : Hachette.

Lo, B. Watson, R., Comber, T. (1996). Achieving Balance in Software Engineering Curricula. IEEE.

Mace, G. (1988). *Guide d'élaboration d'un projet de recherche.* Québec : Les Presses de l'Université Laval.

Maibaum, T.S.E. (2000). Mathematical foundation of software engineering. *Proceedings of the Conference on the Futur of Software Engineering.* ACM, 161-172.

Mayall, W.H. (1979). *Principles in Design.* New York : Van Nostrand Rienhold.

McBreen, P. (2002). *Software Craftmanship,* The New Imperative. New York : Addison-Wesley.

McConnell, S. (1997). Software's Ten Essentials. *IEEE Software,* Mars/Avril, Vol 14, No 2, 143-144.

McConnell, S. (1999). Software Engineering Principles. *IEEE Software,* Mars/Avril, Vol 16, No 2, 6-8.

Meyer, B. (2001). Software Engineering in the Academy. *IEEE Computer,* Mai, 28-36.

Mills, H.D. (1980). The management of software engineering : Part I : Principles of software engineering. *IBM Systems Journal.* Vol 19, No 4, 414-420.

Moore, J.W. (1998). Software Enginnering Standards – A user's Road Map. Los Alamitos: IEEE Computer Society Press.

Moore, J.W. (2006). The Road Map to Software Enginnering – A Strandards-Based Guide. Los Alamitos : IEEE Computer Society, Wiley-Interscience.

Myers, J.P. (2001). Software Engineering Throughout a Traditional Computer Science Curriculum. *The Journal of Computing in Small Colleges*, Janvier, Vol.16 , No 2, 31-41.

Nadeau. R. (1999). Vocabulaire technique et analytique de l'épistémologie. Collection Premier Cycle. Paris : Presses Universitaires de France.

Naur, P., Randell B. (1968). Proceedings of the NATO Conference on Software Engineering, Garmish, Octobre. Germany : NATO Science Committee.

Neumann, P.G. (1993). The Role of Software Engineering. *Communications of the ACM,* Mai, Vol 36, No 5, 114.

Parnas, D.L. (1997). Software Engineering : An Inconsummated Marriage. *Communications of the ACM*, Novembre ,Vol 40, no 9, 128.

Pierce, K.R. (1997). Teaching Software Engineering Principles using Maintenance-based Projects. 10th Conference on Software Engineering Education and Training (CSEET '97). IEEE.

Project Management Institute. (2004). *A Guide to the Project Management Knowledge.* Édition 2004. Newport Square : http://www.pmi.org.

Ralston, A., Reilly, E., Hemmendinger, D, (2000). *Encyclopedia of computer science*, (4e éd.). New York : Grove's Dictionaries.

Robert, P. (2002). *Le petit Robert.* Paris : Dictionnaires Le Robert.

Robert, P. (2001). Le grand Robert. Paris : Dictionnaires Le Robert.

Rodgers, G.F.C. (1983). *The Nature of Enginnering.* London : The MacMillan Press LTD.

Ross,D.T., Goodenough,C.A., Irvine, C.A., Softech Inc. (1975). Software Engineering: Process, Principles, and Goals. *Computer,* Mai, 17-27.

Royce, W. (1970). Managing the development of Large Software Systems. *Reprinted in 9th Internaltional Conference on Software Engineering, IEEE Compuetr Society Press*, 328-338.

Séguin, N. (2001). Étude du principe fondamental -Investir dans la compréhension du problème. Rapport technique. ETS.

Séguin, N., Loiselle-Dupuis, R., Abran, A., Dupuis, R. (2006). Principes fondamentaux du génie logiciel et logiciel libre. *Colloque sur le logiciel libre en tant que modèle d'innovation sociotechnique*. ACFAS.

Shaw, M. (1990). Prospects for an Engineering Discipline of Software. *IEEE Software*, Novembre, 15-24.

Simpson, J., Weiner, E. (1989). *The Oxford English Dictionary.* (2e éd.). Oxford : Clarendon Press.

Sitaraman, M. Gray, J. (1993). Software Reuse : a contexte for introducing software engineering principles in a traditional computer science second course. ACM.

Sobel, A. (2002). Computing Curricula – Software Engineering Volume, *IEEE Computer Society et ACM Education Board*, Fisrt Draf, Aout 28.

Sommerville, I. (2001). *Software Engineering.* (6^{e} éd.), England : Addison-Wesley.

Sparkman, T.G. (1999). Lessons Learned Applying Software Engineering Principles to Visual Programmaing Language Application Development. *COMPSAC 99.* IEEE.

Taylor, P. (2001). Interpreting Mayall's 'Principles in Design', Proceedings Software Engineering Conference, pp. 297-305.

Thomas, R., Semeczko, G., Morarji, H., Mohay, G. (1994). Core Software Engineering Subjects : A Case Study ('86 – '94). Software Education Conference SRIG ET '94. IEEE Computer Society. 133-140.

Tomayko, J.E. (2000). A Historian's View of Software Engineering. *Thirteenth Conference on Software Engineering Education and Training*, Mars 6-8, 101-108.

Vincenti, W.G., (1990). *What Engineers Know and How They Know it*, Baltimore and London : Johns Hopkins.

Vliet, H.V. (2000). Software Engineering – Principles and Practice. (2^{e} éd.). Toronto : Wiley.

Wasserman, A.I. (1996). Toward a Discipline of Software Engineering. IEEE Software, Novembre, 23-31.

Webster's dictionary of English usage. (1989). Springfield, Ma : Merriam-Webster.

Wiegers, K.E. (1996). *Creating a Software Engineering Culture*. New-York : Dorset House Publishing.