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.

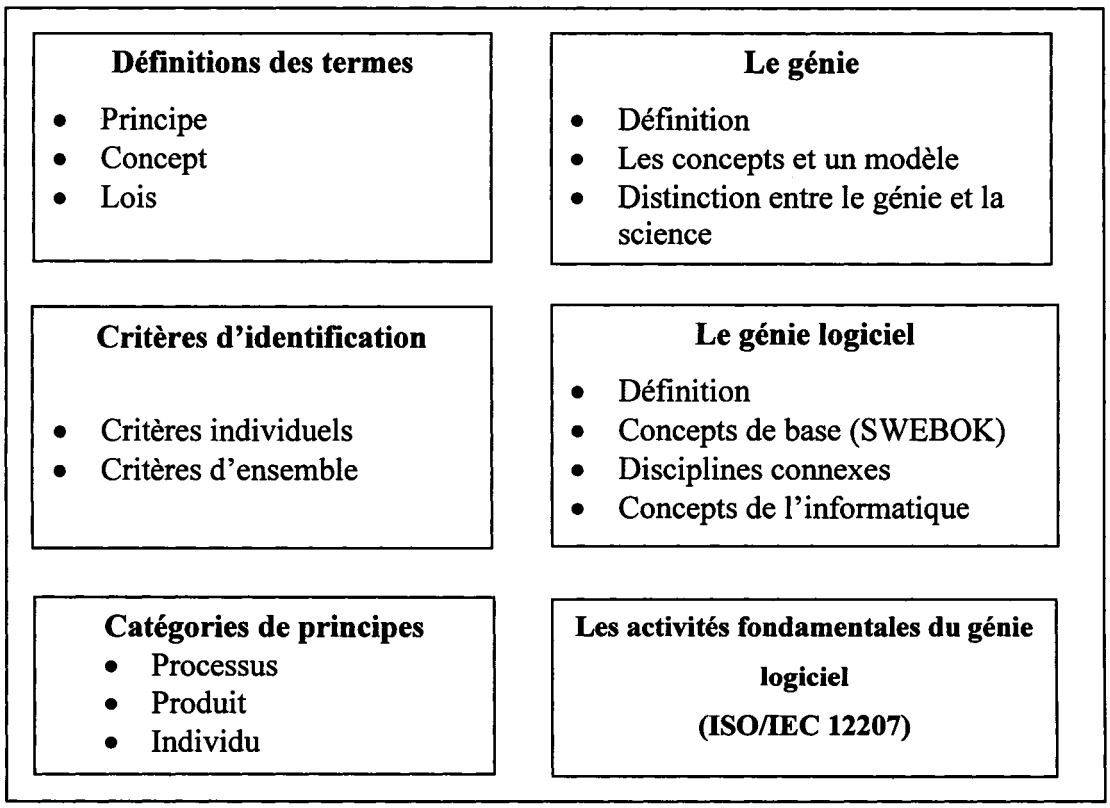

Figure 23 Éléments du cadre conceptuel d'analyse

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.

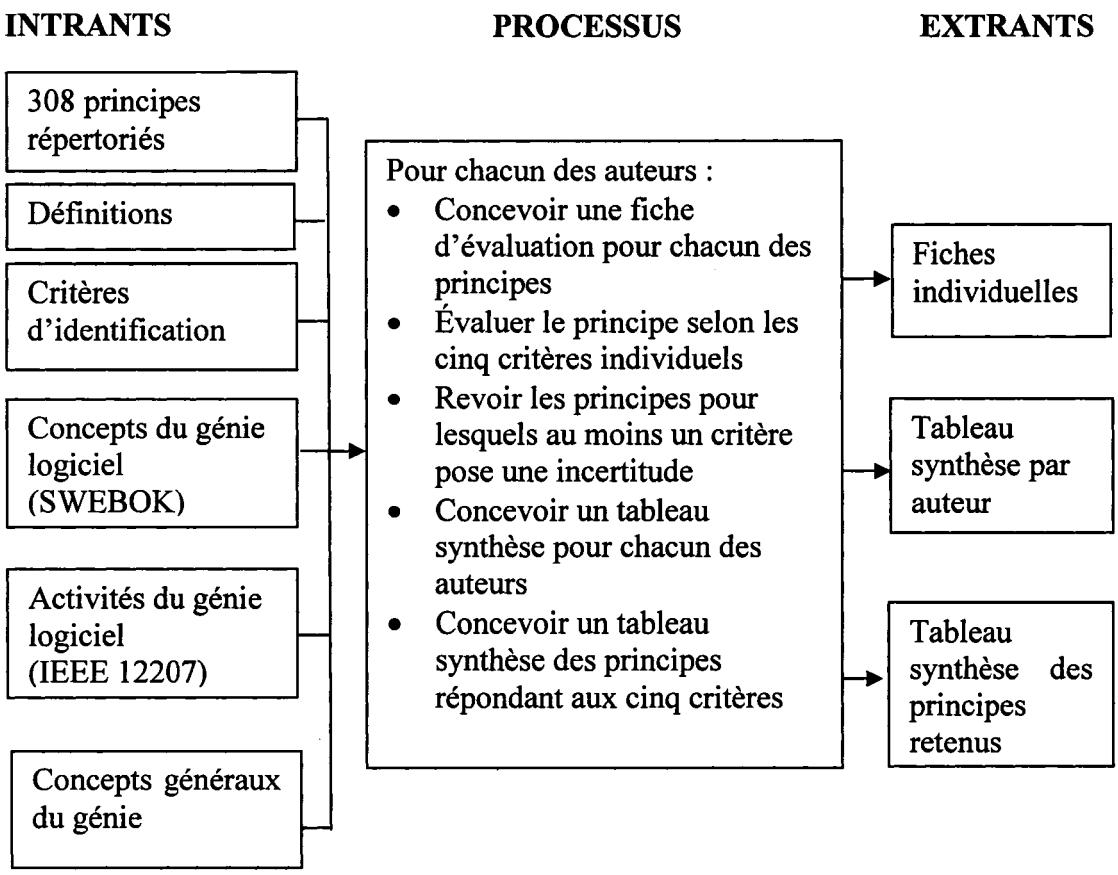

Figure 24 Méthode de recherche de la phase 2

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