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).

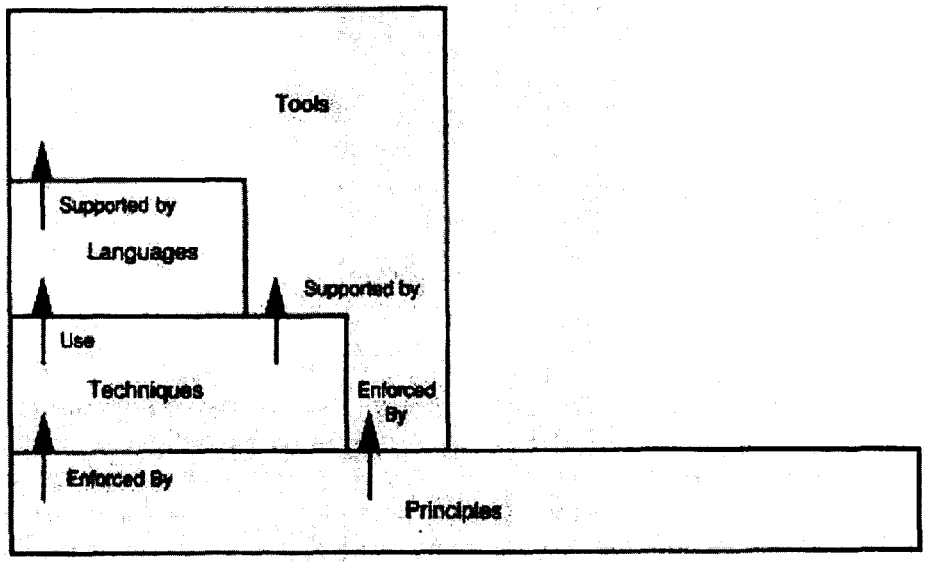

Figure 5 Influence des principes (Davis 1995 p. 4)

[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).

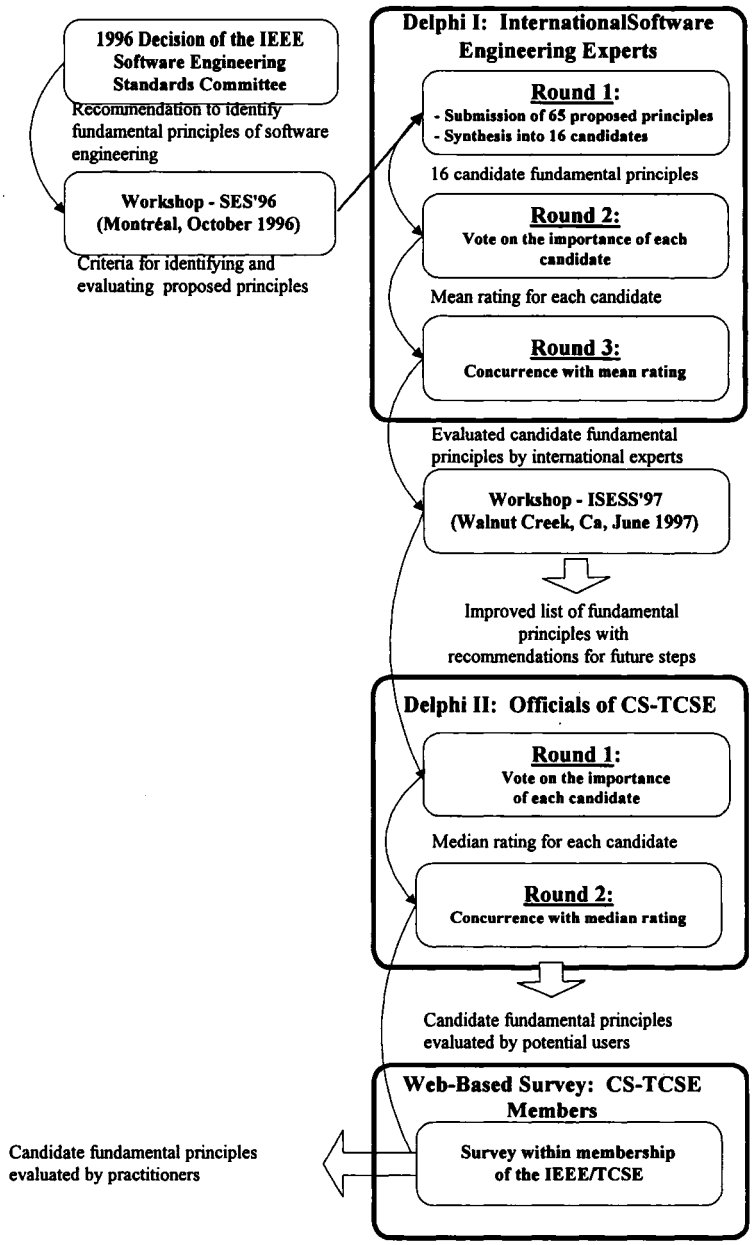

Figure 6 Démarche méthodologique de l'étude de Bourque et al. (2002)

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.