Les principes du génie logiciel s'appliquent–ils au logiciel libre ?

Normand Séguin, UQAM

Renaud Loiselle Dupuis, UQAM

Alain Abran, ETS

Robert Dupuis, UQAM

Abstract
L'identification des principes du génie logiciel est un des thèmes de recherche de l'équipe. Après avoir recensé plus de 300 propositions, le travail d'analyse, en 3 étapes, a permis de réduire à 39 le nombre de principes. Dans cet article, les auteurs désirent vérifier quels sont les principes du génie logiciel qui peuvent s'appliquer au contexte d'un projet complexe et d'envergure tel Mozilla. Nous identifions cinq principes qui s'appliquent fortement au projet, ainsi que ceux qui ne s'appliquent pas du tout et même qui sont contredits par l'approche du open source.

Introduction

Depuis sa naissance à la fin des années soixante, le génie logiciel a connu beaucoup de développement et d'évolution. Les outils de développements, les méthodes et les techniques du génie logiciel évoluent rapidement et ont un cycle de vie très court. Ainsi, lorsque les technologies deviennent désuètes, les méthodes et les techniques associées deviennent aussi désuètes. Ghezzi et al. (2003) soulignent que la discipline ne peut se baser sur les méthodes, les techniques et les outils pour établir ses fondements. La discipline doit plutôt s'appuyer sur un ensemble de principes fondamentaux qui seraient plus durables et moins sensibles au court cycle de vie des technologies. Depuis 2001, nous avons entrepris de mieux cerner le sujet des principes fondamentaux du génie logiciel. Plus de 300 propositions de principes ont été recensés dans la littérature du début des années soixante-dix jusqu'aujourd'hui (Abran 2004). Cependant, la discipline du génie logiciel ne peut comporter autant de principes fondamentaux. Notre groupe de recherche a donc entrepris d'analyser cette banque de principes d'une façon méthodique. Cet article présente brièvement un sommaire des étapes de la méthode suivie, les résultats jusqu'à ce jour de nos travaux et nous identifions les principes qui s'appliquent fortement au contexte du logiciel libre, plus précisément dans le cadre du projet Mozilla.

Objectif de la recherche

La revue de la littérature sur les principes fondamentaux du génie logiciel a mis en lumière, en premier lieu, un grand nombre de principes. L'étude de la revue de littérature a aussi permis de constater une faiblesse importante au niveau des méthodes d'identification des principes, quand méthode il y a. En effet, plusieurs auteurs n'utilisent aucune méthode, ni aucun critère d'identification pour soutenir les principes qu'ils proposent. De plus, rares sont les auteurs qui ont proposé une définition du terme principe. De ce fait, nous avons constaté une grande confusion dans l'utilisation des termes : principes, concepts et techniques.

Un des objectifs principaux de la recherche est d'analyser la banque de principes recensés à l'aide d'une méthode structurée afin de déterminer quelles sont les propositions qui seraient

vraiment des principes. Dans cet article, nous nous concentrons particulièrement à identifier, dans les principes retenus, lesquels s'appliquent au contexte particulier du logiciel libre.

Qu'est-ce qu'un principe?

Le terme *principe* trouve ses origines latines dans le terme *principium* signifiant le commencement ou l'origine. A cet effet, le premier sens donné au terme principe fait référence « *aux origines, causes premières ou élément constituant* ». (Le Robert). Les principes sont à la base d'une discipline, d'un phénomène ou du fonctionnement d'un objet. Un deuxième sens donné au terme principe est « *une proposition première, posée et non-déduite* » (Le Robert, Le Grand Robert). Une proposition est un énoncé qui exprime une « *relation entre deux ou plusieurs concepts* ». Ainsi, un principe serait d'abord une proposition composée de concepts. De ce fait, un concept à lui seul ne pourrait être considéré comme un principe. Une proposition non-déduite signifie qu'un principe ne peut être déduit d'un autre principe. Nous proposons la définition suivante pour le terme principe:
« *Proposition fondamentale de la discipline formulée sous forme prescriptive (règle), à la source des actions, pouvant être vérifiée dans ses conséquences et par l'expérience* »

Critères d'identification des principes

Au niveau de la littérature recensée, seul Boehm (1983) et Bourque et al (2001) ont spécifié des critères d'identification des principes. Considérant la définition retenue d'un principe et le cadre conceptuel d'analyse retenu, nous avons identifié sept critères, cinq individuels et deux d'ensemble. Les critères individuels concernent chacune des propositions prises individuellement, tandis que les deux critères d'ensemble s'appliquent à l'ensemble des propositions. Le tableau 1 présente les critères d'identification des principes.

Critères d'identification individuels
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 une activité du génie logiciel. (adapté de Bourque et al.2001)
3. Le principe ne doit pas énoncer un compromis entre deux actions ou concepts. (Bourque et al. 2001 et de Moore)
4. Un principe du génie logiciel devrait inclure des concepts reliés à la discipline ou au génie. (Bourque et al.2001)
5. La formulation d'un principe doit permettre de le tester en pratique ou de le vérifier dans ses conséquences. (Bourque et al.2001)
Critères d'ensemble
6. Les principes devraient être indépendants (non déduit). (Boehm 1983)
7. Un principe ne doit pas contredire un autre principe connu. (Bourque et al.2001)

Tableau 1 - Liste des critères retenus

Étapes de la méthode

La méthode conçue pour analyser chacun des principes se décompose en trois phases. Nous résumons brièvement les deux premières phases. La première phase applique les cinq critères individuels d'identification à chacun des 308 principes recensés. En plus, les principes ne doivent pas représenter une activité du génie logiciel définie à la norme ISO/IEC 12207. Le guide SWEBOK fournit les principaux concepts du génie logiciel. Un principe doit contenir au moins un concept de la discipline pour être retenu. L'exécution de la phase 1 a permis de retrancher 269 propositions. Celles-ci ne satisfont pas à au moins un des cinq critères individuels d'identification.

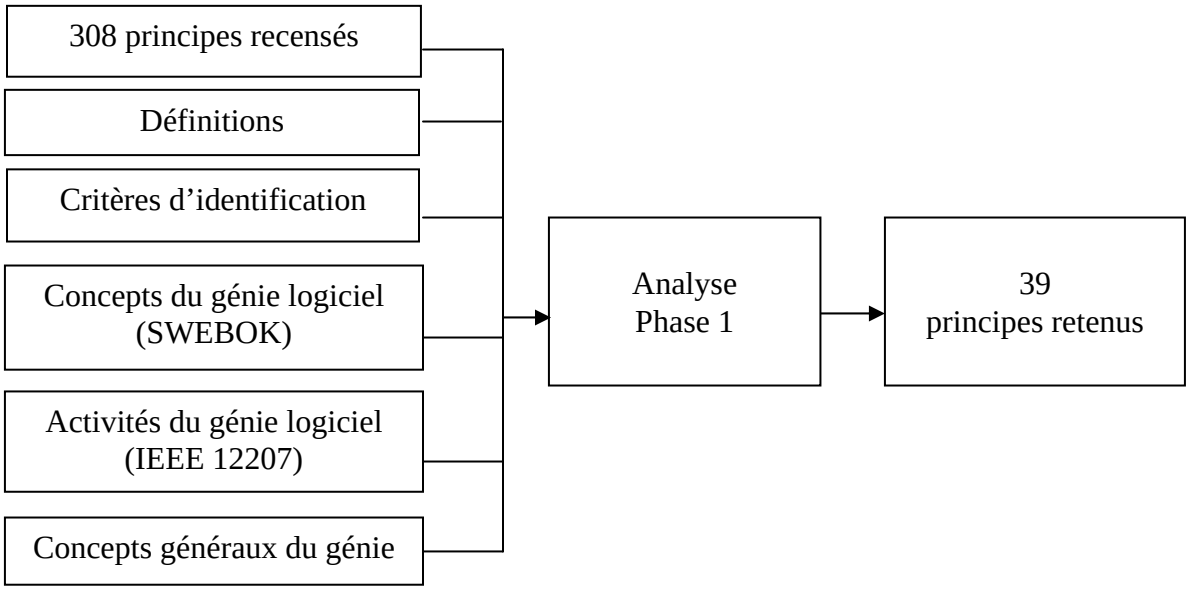

Figure 1 - Éléments d'analyse de la phase 1

La phase 2 applique les deux critères d'ensemble sur les 39 principes retenus à la phase 1. Pour faciliter l'application du critère no.6 à l'effet qu'un principe doit être indépendant et non déduit, les principes sont classés selon trois catégories : processus, produit et individus. Cette classification permet de détecter plus facilement les principes qui peuvent être déduits de d'autres. La phase 2 procède également à l'établissement de liens entre les principes retenus de la catégorie processus et les activités décrites à la norme ISO/IEC 12207. La figure 2 présente les éléments de la deuxième phase d'analyse des principes.

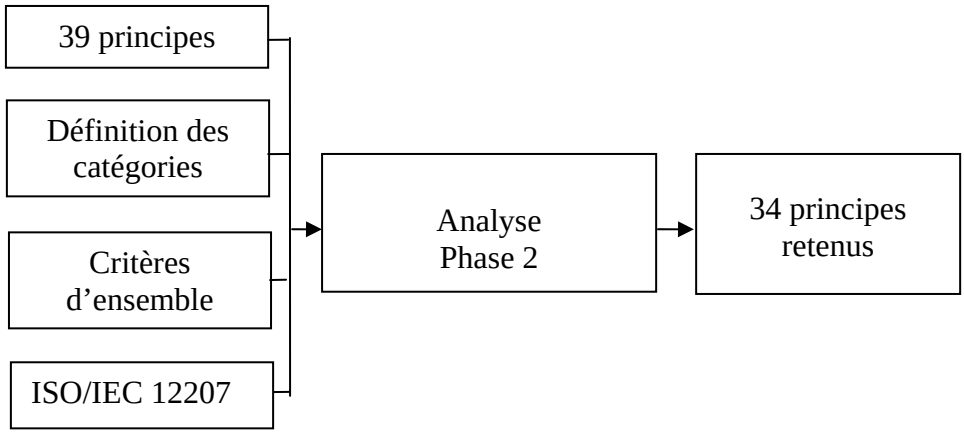

Figure 2 - Éléments d'analyse de la phase 2.

La phase 2 n'a pas détecté de propositions en conflit avec d'autres propositions. Cinq propositions ont été écartées du fait qu'elles pouvaient être déduites de d'autres plus générales. Cependant, nous avons conservé dix propositions qui peuvent être déduites de d'autres, mais dont la formulation ajoute une précision sur l'action à faire. Le tableau 2 présente les propositions retenues suite à la phase 2 de l'analyse. Les propositions marquées d'un astérisque représentent des principes qui peuvent être déduits, mais qui apportent une précision intéressante sur l'action à exécuter.

No.	Proposition
1	Align incentives for developer and customer
2	Apply and use quantitative measurements in decision making
3	Build software so that it needs a short user manual
4	Build with and for reuse
6	Define software artifacts rigorously
9	Design for maintenance*
10	Determine requirements now*
11	Don't overstrain your hardware
13	Don't try to retrofit quality
14	Don't write your own test plans*
15	Establish a software process that provides flexibility
16	Fix requirements specification error now*
17	Give product to customers early*
19	Grow systems incrementally
20	Implement a disciplined approach and improve it continuously
21	Invest in the understanding of the problem
22	Involve the customer
23	Keep design under intellectual control*
24	Maintain clear accountability for results*
25	Produce software in a stepwise fashion*
26	Quality is the top priority; long term productivity is a natural consequence of high quality
27	Rotate (high performer) people through product assurance
28	Since change is inherent to software, plan for it and manage it
29	Since tradeoffs are inherent to software engineering, make them explicit and document it*
30	Strive to have a peer, rather than a customer, find a defect
31	Tailor cost estimation methods
32	To improve design, study previous solutions to similar problems
33	Use better and fewer people
34	Use documentation standards
35	Write programs for people first
36	Know software engineering's techniques before using development tools
37	Select tests based on the likelihood that they will find faults
38	Choose a programming language to assure maintainability
39	In face of unstructured code, rethink the module and redesign it from scratch.*

Tableau 2 - Propositions retenues suite à l'analyse phase 2
*** : propositions déduites, mais conservées**

Principes et le projet Mozilla

Les principes retenus sont intimement associés au contexte du génie logiciel, mais qu'en est-il pour le logiciel libre? Nous avons donc fait l'exercice de croiser notre liste des 34 principes au contexte du projet Mozilla, né d'un « joint-venture » entre Netscape/AOL et Mozilla.org. Nous résumons brièvement les principales caractéristiques du projet (Baker 2006).

Le projet Mozilla est un projet de grande envergure et complexe. Pour arriver à son but, le groupe a mis en place un processus discipliné de développement réparti. Il était de rigueur que le développement du projet ne se fasse pas d'une façon chaotique. Dans ce but, un contrôle à trois volets a été mis en place.

Le premier volet concerne la gestion du code source, le cœur de tout projet *open source*. Le code source est bien sur géré par un outil de gestion du code dont seul l'élite des développeurs peut y entrer les modifications. Ce volet comprend aussi un processus de revue technique à deux niveaux. Le premier niveau concerne la revue d'un fragment de code tel un composant. Chaque développeur doit suivre un guide de programmation. Le deuxième niveau est une revue d'intégration (*super review*) du code dans l'ensemble du logiciel. Seule l'élite des développeurs est autorisée à faire ce type de revue. Le code développé doit satisfaire aux exigences de ces deux niveaux de revue avant d'être intégré à l'ensemble. Les bugs sont aussi documentés au niveau du gestionnaire de sources et des mesures de performances sont consignées.

Le deuxième volet du contrôle touche à la gestion des versions livrables du logiciel. Il y a une planification des versions, une procédure formelle pour assembler une version livrable et en organiser la distribution.

Le troisième volet concerne l'assurance qualité. Tôt dans le projet Mozilla, il a été constaté que le groupe composant le noyau des développeurs ne pouvait être assez diversifié pour tester toutes les possibilités du logiciel en considérant les configurations presque infinies de chacun des futurs utilisateurs. Les responsables ont ainsi choisi de diriger un groupe d'assurance qualité dont les testeurs provenaient de la communauté. Les tests effectués ont été de plus en plus structurés et méthodiques. De plus, les testeurs n'ont pas que rapportés les bugs, mais ils ont aussi procédé à la vérification des bugs rapportés par d'autres afin de voir s'ils étaient réels et d'identifier dans quel contexte ils pouvaient être reproduits. Cette façon de faire aide grandement les développeurs à corriger les problèmes soulevés.

Nous constatons que pour un projet de type *open source*, le projet Mozilla a mis en place un processus discipliné pour mener à terme le développement du logiciel. Comparativement à des projets commerciaux, nous constatons l'importance relativement réduite des exigences logiciel (définition, validation) et des produits intermédiaires issus du processus tel les documents d'analyse, de design et les nombreux plans (de tests, de projet, de risque, etc) normés par l'IEEE par exemple. Le point central du contrôle pour le projet est la gestion du code source.

Cinq principes s'appliquant au projet Mozilla

En considérant les 34 principes retenus, nous avons identifié cinq principes dont l'application est évidente dans le projet Mozilla.

1 – "*Align incentives for developer and customer*"
Dans le domaine du génie logiciel traditionnel, il est fréquent que le management doive motiver les développeurs à respecter les échéanciers en établissant des récompenses ou des pénalités. Au niveau du logiciel libre, les incitatifs existent, mais leur nature est différente. Les incitatifs explicites sont liés au mérite et au niveau de la contribution au projet. Le mérite se récompense par une autorité accrue sur le développement du code. Également, l'accès au gestionnaire du code source se donne également au mérite. Implicitement, plusieurs incitatifs proviennent de la base du mouvement du logiciel libre et sont associés aux motivations fondamentales d'en faire partie. La fierté de participer au projet, la passion et la motivation de voir avancer la cause du logiciel libre sont, entre autres, des incitatifs implicites qui proviennent des développeurs et non imposés par le management. C'est une différence importante par rapport au développement classique.

2 – « *Give product to customer early* »
Au niveau du développement conventionnel, ce principe préconise à l'origine l'utilisation de prototype afin d'obtenir du client une certaine validation des exigences du logiciel. Au niveau du logiciel libre, la communauté s'attend à utiliser des versions préliminaires, mais fonctionnelles du produit et non des prototypes du style coquille vide. La communauté se développe et s'organise rapidement autour d'une telle version préliminaire. Le cycle de rétroaction entre développeurs et utilisateurs peut ainsi s'enclencher par l'entremise de canaux publics de communication tels les listes de distribution, les forums et le suivi des problèmes.

3 – "*Implement a disciplined approach and improve it continuously*"
Dans le cadre du projet Mozilla, la première partie de ce principe a été appliquée. Comme nous l'avons décrit antérieurement, un processus discipliné de gestion du code source, des versions livrables et de l'assurance qualité a été mis en place. Il est à noter que certains contributeurs au projet jugeaient ce processus discipliné plutôt comme de la bureaucratie. La deuxième partie du principe ne semble pas s'appliquer cependant. Le but premier est d'avoir un processus suffisamment discipliné pour éviter le chaos et de mettre l'énergie sur le développement du produit. Ainsi, il ne semble pas y avoir de besoin d'optimiser le processus d'une façon continuelle.

4 – "*Involve the customer*"
Dans le développement classique, il est suggéré que le client soit impliqué dans les différentes étapes du processus de développement afin qu'il puisse aider à valider l'avancement du projet. Au niveau du logiciel libre, le client peut être remplacé par la communauté gravitant autour du produit. Cette communauté a aussi plus de pouvoir sur l'évolution du produit qu'un simple client. Dans ce sens, la communauté est une des grandes forces du développement du logiciel libre. Dans le cadre du projet Mozilla, la communauté a joué un rôle de premier plan, entre autres, dans la phase de test du logiciel. La communauté représente la diversité recherchée par les développeurs qu'aucune entreprise ne peut se permette. Selon le modèle *open source*, la

communauté a différents canaux de communication pour échanger. Les listes de distribution, les forums de discussions, l'accès à un système public de gestion des bugs, sont entre autres, des moyens d'impliquer la communauté dans l'évolution du produit.

5- "*Strive to have a peer, rather than a customer find a defect"*
Ce principe s'applique très bien au contexte du logiciel libre en général et au projet Mozilla en particulier. Comme le souligne Baker : « …*open source projects rely on peer review*.. ». Les membres de la communauté intéressés à tester le logiciel offrent une contribution importante dans la détection des bugs. Dans le cas du projet Mozilla, une partie de la communauté s'est organisée autour d'un leader, membre du noyau de l'organisation pour effectuer des tests structurés et méthodiques du logiciel. De plus, les membres ont vérifié les bugs rapportés afin de mieux cibler l'origine et dans quel contexte particulier ils se produisent. Ce travail a permis aux développeurs de procéder rapidement aux correctifs.

Les principes moins pertinents au mouvement open source

Nous soulignons que certains principes ne trouvent pas d'écho explicite au contexte du logiciel libre. A titre d'exemple, les principes touchant les exigences du logiciel (no.10 et 16) ne se reflètent pas explicitement dans le projet Mozilla. Il est de même pour les principes qui touchent à la maintenance et aux changements (no.9, 28 et 38). Le principe no.9 reflète typiquement une particularité du génie à l'effet de prévoir dès le design, les activités de maintenance et de les faciliter. Nous constatons aussi que les principes (no.24 et 31) qui touchent à la gestion et à l'estimation des coûts ne sont pas une priorité du mouvement *open source*. Dans un contexte traditionnel de développement, le management doit exercer un contrôle sur les coûts, les délais et la qualité afin de respecter le contrat avec le client et espérer faire des profits. La qualité est présente dans le projet Mozilla, mais ne constitue la priorité la plus élevée, comme le souligne le principe no.26. Nous observons que le principe no.33 (*Use fewer and better people*) est même contredit par le mouvement *open source*. En effet, le mouvement va plutôt favoriser le contraire afin d'obtenir une grande diversité de contributeurs au projet.

No.	Proposition
9	Design for maintenance*
10	Determine requirements now*
16	Fix requirements specification error now*
24	Maintain clear accountability for results*
26	Quality is the top priority; long term productivity is a natural consequence of high quality
28	Since change is inherent to software, plan for it and manage it
31	Tailor cost estimation methods
33	Use better and fewer people
38	Choose a programming language to assure maintainability

Tableau 3 - Principes moins pertinents au mouvement open source

Conclusion

A la base de cette première analyse, nous voulions vérifier quels principes du génie logiciel pouvaient s'appliquer au projet complexe *open source* Mozilla. Par contre, nous observons que

certains principes ne sont pas pertinents au contexte de ce projet. Particulièrement, les principes reliés aux exigences du logiciel, à la maintenance et l'estimation des coûts. Nous avons observé qu'un principe est contredit par le fondement même du mouvement du logiciel *open source*. Nous n'avons pas traité des principes relativement neutres face au projet Mozilla. A cet effet, il faudrait étudier d'autres projets de l'envergure de Mozilla pour tirer des conclusions définitives.

Références

Abran, Alain; Séguin, Normand; Bourque, Pierre; Dupuis, Robert, **The Search for Software Engineering Principles: An Overview of Results** , Conference sur "Principles of Software Engineering - PRISE 2004" , Buenos Aires, Argentine , 2004 , pp. 51-60 .

Baker, Mitchell, **The Mozilla Project: Past and Future**, Open Source 2.0, O'Reilly, 2006

Boehm, B.W., **Seven Basic Principles of Software Engineering.** *The Journal of Systems and Software*, Vol.3, no 1, Mai, 366-371, 1983.

Bourque, P.; Dupuis, R.; Abran, A.; Moore, J.W.; Tripp, L.L.; Wolff, S. **Fundamental Principles of Software Engineering - A Journey**. *Journal of Systems and Software*, 2002.

Chris DeBona, D. Cooper et M. Stone, éd., *open sources 2.0*, O'Reilly, 2006.

Ghezzi, C., Jazayeri, M., Mandrioli, D.. ***Fundamentals of Software Engineering***. (2e éd.). New-Jersey : Prentice Hall, 2003.

Moore, J.W., **Software Enginnering Standards – A user's Road Map**. IEEE Computer Society, 1998.

www.mozilla.org