UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ANALYSE, ÉVALUATION ET AMÉLIORATION DE LA PERFORMANCE DU PROCESSUS DE DÉVELOPPEMENT LIBRE : UNE APPROCHE PAR LA NORME ISO/IEC 29110

MÉMOIRE PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

ADÈLE LARISSA NSOA

MAI 2012

REMERCIEMENTS

Je remercie premièrement le professeur Normand Séguin de m'avoir invité à poursuivre mon projet de maîtrise sous sa direction. Il a été vraiment disponible pour moi tout au long du déroulement de ce projet de recherche et m'a procuré de précieux conseils et sa vision globale du projet a guidé et soutenu ma démarche.

Je tiens à remercier les professeurs qui accepteront d'évaluer le rapport de mon travail de recherche.

Je tiens à remercier particulièrement mon ami Edouard Fonh-Gbei pour avoir cru en moi et pour m'avoir soutenu tout au long de mon année académique.

Je remercie également ma famille toute entière: ma mère, mon père, ma sœur et mes frères, pour m'avoir apporté un soutien moral inestimable et d'avoir cru en mes capacités et en ma volonté de terminer ce projet de recherche.

Je n'oublie pas de remercier mon amie Amina Elbekkali pour ses encouragements.

TABLE DES MATIÈRES

LISTE DES FIGURES

Figure **Page**

LISTE DES TABLEAUX

Tableau **Page**

LISTE DES SIGLES, ABRÉVATIONS ET ACRONYMES

AGPL	Affero General Public License. Généralement connue sous le nom de GNU AGPL
AN	Analyste
BSD	Berkeley Software Distribution License
CCB	System control board
CUS	Client
CVS	Concurrent Versions System
DES	Designer
EMO	Eclipe Management Organization
EMO(ED)	Sous-ensemble de l'EMO: En fait c'est un directeur exécutif (il/elle) qui peut déléguer une autorité spécifique à une autre personne.
Ezmlm	Progiciel pour la gestion des listes de diffusion électroniques
FDL	Free Documentation License. Généralement connue sous le nom de GNU FDL
F/OSS	Free / Open Source Software
FSF	Free Software Foundation
GNATS	GNU GNATS est un ensemble d'outils pour le suivi des bogues rapportés par les utilisateurs vers un site central.

GNU	Système d'exploitation libre, initié en 1983 par Richard Stallman dans le projet GNU [101]. Il est généralement aussi connu sous le nom de GNU/Linux.
GPL	General Public License. Généralement connue sous le nom de GNU GPL
IEEE	Institute of Electrical and Electronics Engineers
IRC	Internet Relay Chat
ISO	International Organization of Standardization
LGPL	Lesser General Public License. Généralement connue sous le nom de GNU LGPL
LXR	Linux Cross Referencer. Utilisé pour l'indexation de code source, particulièrement dans de gros projets tel que le noyau linux
Mozmill	Outil de test d'interface utilisateur pour les applications basées sur Mozilla. Il est utilisé par la Mozilla Corporation et Mozilla Messaging pour exécuter des tests fonctionnels par rapport à Firefox et Thunderbird.
OSI	Open Source Initiative
PM	Projet Manager
PMC	Projet Managment Committee
PR	Programmeur
SVN	Apache Subversion. C'est un logiciel de gestion de versions.
TinderBox	Outil supportant les tests de logiciel
TL	Technical lead ou chef technique en français
ViewCVS	Outil permettant de visualiser les dépôts CVS ou SVN via un navigateur web
WT	Groupe de travail

RÉSUMÉ

Les F/OSS[1] font face à de nombreux problèmes de qualité [67, 108, 115] et cette problématique de qualité est d'actualité. Malgré un nombre important de publications sur les F/OSS, nous déplorons la rareté des recherches qui se sont penchées sur l'application des normes de base de développement du génie logiciel au processus de développement libre. Cependant, certains projets libre ont pu développer du logiciel de qualité et ont maintenu ce niveau de qualité dans les logiciels développés ou maintenus au sein de leurs communautés. Cette recherche a donc pour but d'analyser si une norme de développement du génie logiciel telle que la jeune norme ISO/IEC 29110 peut soutenir ou améliorer le processus de développement libre compte tenu de la culture du libre.

La méthodologie de recherche utilisée est divisée en deux phases. La première phase est une étude exploratoire des F/OSS au travers de l'étude des projets : Linux, Apache, Mozilla et GNOME. Elle a permis de se familiariser aux F/OSS, d'identifier les problèmes de qualité rencontrés, d'extraire et d'évaluer le processus de développement actuel des F/OSS par rapport à la norme ISO/IEC 29110. Les résultats de cette évaluation rejoignent à littérature en ce sens que le développement libre souffre d'un manque de documentation. La deuxième phase consiste en une analyse de compatibilité des activités du processus d'implémentation de la norme ISO/IEC 29110 avec les F/OSS, afin d'identifier les activités de la norme qui sont compatibles aux F/OSS. Cette évaluation a permit l'identification de 4 activités de la norme ISO/IEC 29110 qui sont compatibles aux F/OSS, et qui ont été la base d'une stratégie d'amélioration de la performance du processus de développement libre.

Mots clés : Assurance qualité, qualité, processus de développement, valeurs, logiciels libre, valeurs, culture, principes, activités, artefacts, norme de développement du génie logiciel, logiciels open source, génie logiciel.

---

[1] F/OSS: *Free / Open Source Software.* Le sigle F/OSS englobe à la fois les logiciels libres et les logiciels open source. Il désigne les logiciels développés ou maintenus sous une licence libre ou open source.

CHAPITRE I

INTRODUCTION

1.1 CONTEXTE DES LOGICIELS LIBRES

Les communautés ou projets de logiciels libres, sont habituellement des organisations hybrides qui combinent le but non lucratif et des stratégies de marché. Elles sont constituées d'une multitude de participants, généralement des bénévoles, distribués géographiquement à travers le monde [92]. Leur principal moyen de communication est entièrement basé sur l'Internet, par le biais d'une infrastructure solide soutenue par de multiples outils qui sont majoritairement des logiciels libres (voir chapitre 3 - section 3.2.4.2.3 pour les détails).

Les logiciels développés ou maintenus dans les communautés de logiciels libres sont de types :

- **Les systèmes d'exploitation :** par exemple, GNU/Linux, Free/Net/OpenBSD [8], Solaris;
- **Les applications web et applications réseau :** par exemple, Mozilla, Apache, Samba;
- **Les interfaces utilisateurs et graphiques :** par exemple, le projet GNOME, Gimp;
- **Les suites Office et graphiques :** par exemple, OpenOffice.org;

- **Compilateurs, éditeurs de programmes et langages de programmations** : GCC, Php, Java, Notepad++, Emacs, Gummi.

Ces exemples confirment simplement les identifications d'Eric Raymond [102] et de Brian Behlendorf [34]. Ces pionniers dans les logiciels libres, ont identifié en 1999 que la majorité des applications de F/OSS sont dans le domaine Internet ou dans les applications systèmes. D'amples exemples de projets de F/OSS[2] sont vus dans certains travaux [12, 70, 77].

Les F/OSS sont divisés en différents sous-projets. Chacun de ces sous-projets possède une communauté de membres qui lui est propre, un portail web, une liste de diffusion des développeurs, un groupe de mainteneurs du projet et dans certains cas un groupe d'assurance qualité. Il n'existe pas réellement une durée maximale de réalisation pour ces projets. Cependant, chaque projet évolue itérativement, version après version [102].

Le mode de développement des F/OSS, bien qu'intriguant et très peu orthodoxe comparativement au développement commercial qui est soutenu par des normes de l'industrie, suscite un intérêt grandissant tant pour de simples utilisateurs que pour des entreprises commerciales. Il offre d'énormes avantages économiques et permet d'aboutir dans certains cas à des logiciels de qualité élevée et très sécuritaires. Certains F/OSS se sont démarqués des autres, c'est le cas des communautés Linux, Mozilla, GNOME, Apache et même Eclipse.

---

[2] F/OSS: *Free / Open Source Software.* Le sigle F/OSS englobe à la fois les logiciels libres et les logiciels open source. Nous l'utiliserons tout au long de notre étude. Généralement dans cette étude, lorsque nous utiliserons «logiciels libres» ou «logiciels open source», nous penserons au sigle F/OSS. La différence entre logiciels libres et logiciels open source sera présentée au chapitre 3.

1.2 CONTEXTE DE L'ÉTUDE

L'utilisation de logiciels libres, depuis leur création jusqu'à présent, est de plus en plus grandissante [13, 100, 103]. Les F/OSS sont une alternative viable face aux logiciels commerciaux [60], de par leurs performances. Dans certains cas, ils sont d'une qualité et fiabilité élevées [26]. Ce contexte très intéressant a suscité ma curiosité face aux logiciels libres. Et donc, il m'a mené à la question spécifique **de voir si une norme de base sur les processus de développement comme la ISO/IEC 29110 pouvait s'appliquer au F/OSS ou en partie aider à améliorer le développement de logiciels libres** (voir chapitre 2 - section 2.3 pour les détails).

Le sujet de mon mémoire me permettra non seulement de démystifier le concept de logiciel libre, mais également de mieux comprendre leur processus de développement.

Ce sujet de recherche présente :

- Un aspect théorique : consistant en une présentation détaillée des valeurs, des pratiques et des problèmes de qualité, ainsi que du mode de développement de logiciel libre;
- Un aspect pratique : qui consiste principalement en deux évaluations. La première consiste à l'évaluation des processus de développement des logiciels libres par rapport à la norme ISO/IEC 29110 pour relever les différences entre les logiciels libres et le développement classique de logiciels. La seconde consiste quant à elle à l'évaluation de la norme ISO/IEC 29110 par rapport aux logiciels libres, question de déterminer les activités de la norme ISO/IEC 29110 qui sont compatibles et ceux non compatibles au contexte particulier du logiciel libre. Cet aspect de compatibilité de la norme ISO/IEC 29110 aux F/OSS va soulever la problématique d'appliquer cette norme au contexte

particulier des F/OSS. Les activités de la norme ISO/IEC 29110 qui seront compatibles au contexte des F/OSS, constitueront la base de notre proposition d'amélioration des performances du processus de développement des F/OSS via notre application de la norme ISO/IEC 29110 aux F/OSS.

1.3 CONTRIBUTIONS

Les contributions essentielles du présent rapport sont :

- L'identification des pratiques et problèmes de qualité des F/OSS;
- L'identification des différences entre le processus de développement de logiciels libres et le processus de développement de logiciels en génie logiciel;
- L'identification des processus ou activités de la norme ISO/IEC 29110 qui sont compatibles aux logiciels libres, et donc qui peuvent s'appliquer aux processus libres;
- La proposition d'une application de la norme ISO/IEC 29110 au contexte des F/OSS.

1.4 STRUCTURE DU DOCUMENT

Le présent document est constitué de six chapitres, sans compter les chapitres pour l'introduction et la conclusion. Au chapitre 2, nous présenterons notre problématique de recherche ainsi que les objectifs que nous devons atteindre à la fin de cette étude. Par ailleurs, le chapitre 2 contient également des définitions des termes clés, utilisés fréquemment dans ce document et qui sont nécessaires à la compréhension de la suite de cette étude. Au chapitre 3, nous présentons une revue de littérature sur les logiciels libres et sur l'assurance qualité en logiciel libre. À la suite de cette revue de littérature, notamment au chapitre 4, nous définissons notre méthodologie de

recherche. Nous procèderons au chapitre 5 à une première analyse du processus de développement de logiciel en nous basant sur l'observation faite, tirée de la revue de littérature du chapitre 3. À la suite de ce chapitre, principalement au chapitre 6, nous introduirons la norme ISO/IEC 29110 par une brève description de son contenu et présenterons pour finir deux évaluations de processus : l'évaluation du processus de développement des logiciels libres par rapport au développement classique de logiciels et l'évaluation de la norme ISO/IEC 29110 par rapport aux logiciels libres. Pour finir, nous présenterons une analyse et les réflexions faites sur les résultats obtenus au chapitre 6. Dans la conclusion, nous répondrons à la question principale de notre sujet de recherche, évaluerons si les objectifs de cette recherche (voir chapitre 2 - section 2.3) sont atteints et présenterons les limites de cette études ainsi que nos travaux futurs.

CHAPITRE II

PROBLÉMATIQUE ET OBJECTIFS

2.1 INTRODUCTION

Dans ce chapitre, nous présentons la problématique de notre recherche ainsi que les objectifs que nous comptons atteindre.

Avant d'aller plus loin, nous définissons certains termes clés que nous jugeons importants pour la suite de notre étude : « principes », « philosophie », « culture », « valeurs ». Ces derniers ont de légères différences d'une communauté à une autre. Nous définissons également les expressions suivantes: « génie logiciel », «assurance qualité » et « ISO /IEC 29110 ».

✓ **Philosophie :**

Provenant du mot grec *philosophia*, le terme en son sens le plus large consiste en l'exercice systématique de la pensée et de la réflexion. Selon Larousse [17], l'une des définitions de la philosophie consiste en : « un système d'idées qui se propose de dégager les principes fondamentaux d'une discipline ». Une autre définition proposée par Larousse [17] est : « la manière de voir, de comprendre, d'interpréter le monde, les choses de la vie, qui guide le comportement ». Donc, si nous associons ces deux définitions au développement, le terme pourrait s'appréhender à la façon ou **le**

comment se fait le développement et permet de ce fait de dégager les principes fondamentaux de ce développement. En ce sens, nous pouvons donc voir le terme philosophie comme un modèle de développement de logiciel. Cette façon de la voir s'appréhende bien à la philosophie dans les logiciels libres.

✓ **Culture :**

Les projets de logiciel libre et open source sont organisés sous forme d'une organisation virtuelle, permettant de gérer et de maintenir le projet. Magaret Elliot [48] définit la culture dans les projets de F/OSS de la façon suivante :

« *Culture is treated as a metaphor for organization, not as a discrete variable within an organization. The virtual community is the organizational culture* » [48].

Nous notons une autre définition de la culture, ici au sens du génie logiciel. La culture est définie par Karl Weigers [121] de la manière suivante :

« *Culture includes a set of shared values, goals, and principles that guide the behaviors, activities, priorities, and decisions of a group of people working toward a common objective.* » [121].

La culture en génie logiciel (figure 2.1) vise à améliorer la qualité de produit logiciel, tant au niveau de la gestion (par le biais de la mise en place d'un environnement de développement qui facilite le développement de logiciel de qualité) qu'au niveau du processus de développement proprement dit.

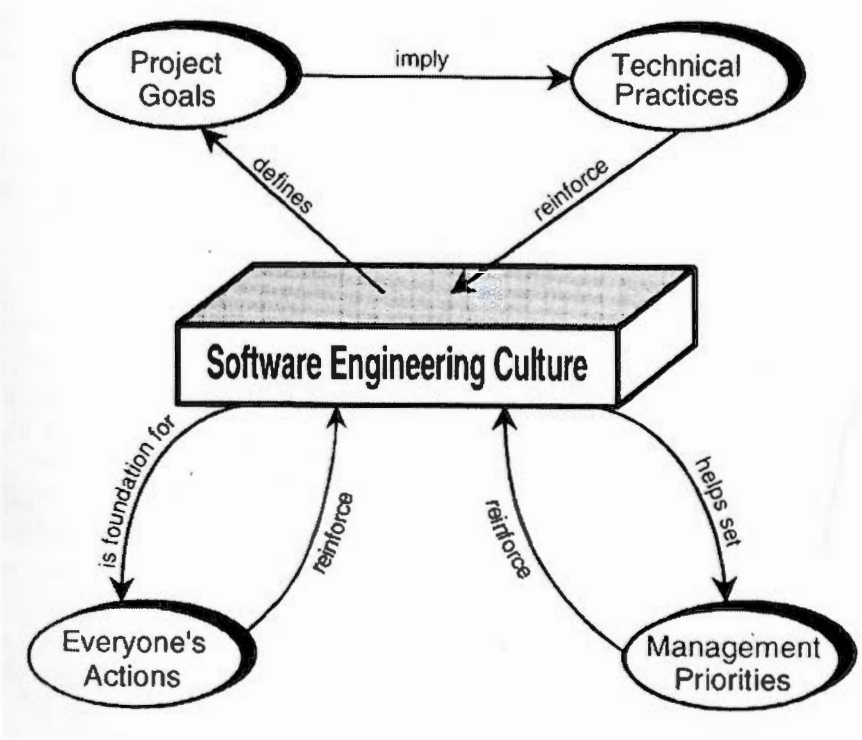

Figure 2.1 Culture en génie logiciel [121]

Le développement d'une culture se base sur l'application des meilleures pratiques de l'industrie, et exige l'application d'une discipline personnelle et professionnelle chez les différents participants aux projets logiciels. Cependant, elle n'élimine pas la créativité chez les développeurs. Dans ce contexte, la culture telle que vu par Weigers pourrait s'adapter au développement du logiciel libre.

✓ **Principe :**

Ce terme est utilisé dans bon nombre de domaines. Il est très souvent vu comme : un fondement, une cause première, une proposition admise comme base d'un raisonnement, une règle générale théorique guidant la conduite [43, 76].

Normand Séguin [109] définit un principe comme suit : « 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 ». C'est cette définition que nous considérons pour notre étude.

✓ **Valeurs :**

Les valeurs sont des règles qui régissent le comportement d'un ensemble de personnes. De manière plus détaillée, « les valeurs représentent des principes auxquels doivent se conformer les manières d'être et d'agir. Ces principes sont ceux qu'une personne ou qu'une collectivité reconnaissent comme idéales [sic] et qui rendent désirables et estimables les êtres ou les conduites auxquelles elles [sic] sont attribuées » [24]. Elles sont implicites dans la culture de chaque société. En appliquant la définition des valeurs précédemment énoncées aux logiciels libres, nous dirons que les valeurs sont des règles implicites à la culture de chaque projet, et qui soutiennent le comportement des membres de ces projets.

✓ **Génie logiciel :**

Selon *IEEE Computer Society* [27], le génie logiciel est: « *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software* ».

✓ **Assurance qualité:**

L'assurance qualité est définie par l'ISO/IEC/IEEE 24765 [72] comme étant : «1) un ensemble d'activités planifiées et systématiques de toutes les actions nécessaires pour fournir l'assurance suffisante qu'un élément produit est conforme aux exigences techniques établies ; 2) un ensemble d'activités destinées à évaluer le processus par lequel les produits sont développés ou fabriqués ; 3) les activités planifiées et systématiques mises en œuvre, dans le système qualité, et démontrées au besoin pour fournir l'assurance suffisante qu'une entité satisfera aux exigences de qualité ». Il est

vraiment important de la garder à l'esprit car c'est cette définition qui est utilisée tout au long de cette étude.

✓ **ISO/IEC 29110:**

C'est une norme développée par l'organisme ISO spécifiquement pour les très petits organismes (TPO) qui développent du logiciel. Les TPO sont des entreprises, des organismes, des départements ou des projets comportant 25 personnes ou moins. La norme ISO/IEC 29110 décrit les phases minimales de développement à la fois du processus de gestion de projet et du processus de mise en œuvre de logiciel, en tenant compte des informations suffisantes pour qu'un TPO puisse décrire sont propre processus de développement [73, 81, 111]. Pour plus de détails, consulter le chapitre 6 et les parties gratuites de la norme ISO/IEC 29110 en ANNEXE D et ANNEXE E .

2.2 PROBLÉMATIQUE

Depuis la création du concept de logiciel libre, face à leur croissance fulgurante [13, 82, 100, 103], un débat est porté sur la question de savoir si les processus de F/OSS peuvent s'appliquer dans d'autres domaines [91]. Sachant que les F/OSS ont réussi où plusieurs entreprises auparavant ont échoué, notamment, le développement de façon distribué via l'Internet [89]. La question soulevée est : quand, où, et de quelle manière le processus de développement de F/OSS se différencie des autres processus [91]? Dans ce contexte, différentes opinions sont nées. Elles opposent les personnes qui critiquent les pratiques utilisées dans les logiciels libres [90] au groupe constitué de praticiens de F/OSS et de chercheurs supportant le mouvement de F/OSS [26, 36, 102, 105]. Ces chercheurs ont noté certaines spécificités dans l'organisation et les processus de F/OSS [26].

À cet effet, Mockus, Fielding et Herbsleb [94] ont eu à identifier des différences entre le développement de F/OSS par rapport au développement traditionnel :

- Les F/OSS sont construits par un nombre souvent élevé de bénévoles;
- Le travail n'est pas assigné aux participants, les personnes entreprennent les travaux qu'elles désirent faire;
- Il n'y a aucune conception explicite au niveau du système, ou même une conception détaillée du système;
- Il n'y a aucun plan de projet, de calendrier, ou liste des livrables.

Les points mentionnés par Mockus, Fielding et Herbsleb [94], traduisent tout simplement la nature non organisée et non-structurée des projets de logiciels libres qui opèrent de façon ad-hoc [104], et donc, au sens classique de développement, sont sujets à l'échec du projet. Cela conduirait à des délais non respectés et à des débordements de coûts (coûts de développement et coûts de main d'œuvre) de projet, car les participants aux F/OSS, qui sont des bénévoles et des experts en leur domaine, ne sont pas payés. Cependant, les considérations en F/OSS, notamment les notions de coûts de développement et des délais, diffèrent de celles du développement traditionnel de logiciel qui nécessitent que les processus soient soigneusement planifiés et organisés. Un autre aspect tout aussi important est que, de plus en plus, certains projets de F/OSS, particulièrement les grands projets, ont recours à une planification dans leur développement. Il s'agit de la planification des « releases » [47, 60], qui est un processus permettant d'aboutir à une version rapide et stable du système logiciel libre. L'usage de la planification des releases dans certains projets libres est en opposition avec l'un des points identifiés par Mockus, Fielding et Herbsleb [94] stipulant que dans le développement libre, « il n'y a aucun plan projet, ou calendrier ».

Bien que la méthodologie de développement de logiciels libres ait permis d'aboutir à des systèmes importants (cas de Linux), dans certains cas de qualité élevée, comparable voire supérieure aux logiciels de développement fermé [26, 67, 108]; les F/OSS souffrent cependant d'un réel **problème de qualité** dans leur développement. Ce problème de qualité, compte tenu de notre observation des F/OSS, serait dû à la fois à un développement distribué géographiquement et au fait que les participants de F/OSS soient des bénévoles. Les recherches de Mockus et ses collaborateurs [94], ainsi que d'autres menées sur le développement de F/OSS [31, 93, 108, 115], conduisent de manière générale à ce problème de qualité dans le développement de F/OSS. Le débat sur la qualité dans les processus de F/OSS est toujours ouvert de nos jours.

Face au problème de qualité dont souffrent les F/OSS, la question est donc de savoir si les processus d'ingénierie peuvent s'appliquer au développement de logiciel libre pour pallier aux lacunes de qualité (voir chapitre 3 pour les détails) que font face bon nombre de projets de type logiciels libres. C'est donc dans ce contexte que porte notre étude, particulièrement sur la question spécifique : **est-ce que la norme ISO/IEC 29110 [73] pourrait soutenir le développement de logiciels libres, compte tenu de la culture et des valeurs sous jacentes au monde du libre?**

2.3 OBJECTIFS

Notre travail est une étude analytique dont le but principal est d'évaluer si la norme ISO/IEC 29110 peut s'appliquer au développement libre. À cet effet, nous présenterons donc : la culture, la philosophie, les valeurs dans le monde du libre, ainsi que leurs pratiques et problèmes de qualité.

Ainsi, afin de mener à bien cette étude, nous nous sommes donc fixés les quatre objectifs suivants :

- ✓ Identifier les problèmes de qualité que rencontrent certains projets de F/OSS;
- ✓ Analyser le processus de développement de logiciels libres :
  - o Déterminer en quoi le développement des logiciels libres se distingue du développement classique, ici, celui en génie logiciel (respect des normes de l'industrie);
  - o D'identifier lesquelles des activités de la norme ISO/IEC 29110 sont incompatibles avec la philosophie de F/OSS et les considérations autour des F/OSS, et lesquelles pourraient améliorer le processus de développement de F/OSS;
- ✓ Présenter la synthèse des résultats obtenus;
- ✓ Présenter les conclusions tirées de cette étude.

Afin de réaliser ces objectifs, nous suivrons notre méthodologie de recherche, laquelle est présentée en détail dans le chapitre 4.

CHAPITRE III

ÉTAT DE L'ART SUR L'ASSURANCE QUALITÉ EN LOGICIELS LIBRES

3.1 LITTÉRATURES SUR LES F/OSS

Depuis des années, les entreprises propriétaires utilisent un mode de développement basé sur des standards de l'industrie, afin d'obtenir des produits d'une certaine qualité, sous des licences protégeant la propriété intellectuelle. Face à toute cette rigueur procédurale et à l'impossibilité des utilisateurs de pouvoir accéder librement au code source, Richard Stallman lance en 1983, le mouvement du « Logiciel libre » et le formalise à travers le projet GNU [101]. Deux années plus tard, Stallman fonde la FSF [54] afin de soutenir et promouvoir le logiciel libre, de même que de défendre les droits ou libertés des utilisateurs. Depuis sa création, le concept de logiciel libre a révolutionné le développement de logiciels et est considéré de nos jours comme un atout important tant pour de simples utilisateurs que pour des entreprises commerciales. Les logiciels libres présentent d'énormes avantages économiques par rapport aux logiciels propriétaires.

Dans la suite de ce chapitre, nous examinerons comment se fait l'assurance qualité dans les projets de F/OSS, ainsi que les méthodes et outils permettant d'aboutir à une qualité élevée du produit logiciel dans les communautés de F/OSS. Bien avant

d'entrer dans le cœur des pratiques des communautés de F/OSS, nous exposerons les valeurs véhiculées dans le développement de F/OSS.

3.1.1 Définition de F/OSS

Tel que définie par le projet GNU [101], un logiciel est dit *libre* si tous les utilisateurs ont les libertés d'exécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel. Cela est plus connu sous l'appellation des **quatre libertés** [101] :

- **Liberté 0**: Liberté d'exécuter le programme pour tous les usages;
- **Liberté 1**: Liberté pour les utilisateurs d'étudier le programme et de l'adapter à leurs besoins. Cette liberté nécessite l'accès au code source;
- **Liberté 2**: Liberté de redistribuer les copies du programme;
- **Liberté 3**: Liberté d'améliorer le programme et de publier vos améliorations pour en faire profiter toute la communauté. Cette liberté requiert également l'accès au code source.

Ces libertés exposent clairement le focus du concept de logiciel libre, qui est : « *La liberté et non le prix* » comme l'a dit Stallman [101]. Cependant, ce concept de « logiciel libre » ne signifie pas être gratuit, car la production et la distribution peuvent être rémunérées mais leur utilisation doit rester libre. Afin que les logiciels dérivés restent libres, en d'autres termes, que les droits d'auteurs restent conservés, les communautés de logiciels libres utilisent des « licences Copyleft[3] ». Cependant, quelque fois dans les communautés de logiciels libres, les logiciels peuvent être « non-copyleftés ». Cela permet ainsi à un utilisateur qui utilise ou qui modifie un

---

[3] Le *copyleft* est une catégorie de licence au sein de la communauté du logiciel libre permettant à un logiciel libre de conserver le statut de logiciel libre à travers ses versions ultérieures. Voir le projet GNU [62, 84, 101] pour plus de détails.

logiciel de type « logiciel libre », de le vendre et de s'en accaparer sous une licence propriétaire (voir tableau 3.1 pour quelques licences en logiciels libres).

Le concept de « logiciel libre » est toute fois confondu avec celui de logiciel *Open source* (confer - ANNEXE A pour les détails de la définition d'un logiciel Open Source tel que défini par OSI [23]). Le concept de logiciel open source émane de celui de « logiciel libre ». Cependant, il se différencie du concept de logiciel libre par sa philosophie. La philosophie liée à la définition du logiciel libre selon Stallman est focalisée sur la liberté tandis que celle des logiciels *Open Source* est un peu plus concrète. L'accent est mis sur les fonctionnalités du logiciel telles que la qualité dans les logiciels Open source [91]. La désignation *Open source* s'applique aux logiciels dont la licence respecte les critères établis par l'*Open Source Initiative* (OSI) [23], alors la désignation *logiciel libre* s'applique aux logiciels dont la licence respecte les critères ayant été formalisés dans le projet GNU [101] et soutenu par la *Free Software Foundation* (FSF) [54]. Tous deux ont en commun les quatre libertés du libre [101] car le concept de logiciel libre tel que vu par Stallman a été à la base de celui de logiciel *Open source*. Néanmoins, du fait que le concept de logiciel *Open source* est régit par une philosophie émanant et partageant celle de logiciel libre [23], les deux mouvements ont donc pratiquement les mêmes : licences, logiciels et le plus important étant des processus de développement grandement semblables [91]. Ainsi, nous engloberons ces concepts dans la suite de notre étude sous forme de F/OSS (*Free / Open Source Software*). Par ailleurs, ces deux concepts (logiciel libre et logiciel open source) n'empêchent pas un individu, ou une entreprise, ou un groupe de personnes, de vendre ou bien de distribuer un logiciel de type libre[4] comme une composante d'un système logiciel qui contient différentes sources. L'appellation

---

[4] Nous considérons qu'un logiciel de type libre est un logiciel développé ou maintenu dans l'une des communautés « logiciel libre » ou « Open source ».

F/OSS nous permettra ainsi de mieux faire ressortir les meilleures pratiques de qualité du monde du libre.

Modèles de licences de F/OSS	Catégories				Compatible avec GPL	Projets
	Copyleftée	Non-copyleftée	Open source	Libre		
GPL	oui	oui	oui	oui	oui	Projet GNU. Exemple : CVS
LGPL	Partiel - lement	oui	oui	oui	oui	Projet GNU. Exemple : La librairie C de GNU
AGPL	oui	oui	oui	oui	oui	Projet GNU
FDL	oui	oui	oui	oui	oui	Projet GNU
BSD	non	oui	oui	oui	non	Apache, Sendmail
MPL/NPL	non	oui	oui	oui	non	Mozilla
EPL	non	oui	oui	oui	non	Eclipse

Tableau 3.1 Quelques licences de F/OSS

3.1.2 Considérations dans les F/OSS

3.1.2.1 Valeurs dans les F/OSS

Les enquêtes menées en 2005 par Michlmayr et ses collaborateurs [92] permettent d'identifier deux caractéristiques ou valeurs essentielles des F/OSS, à savoir :

- Les projets de logiciels libres sont pour la plupart du temps, distribués à travers le monde ;
- Les participants des projets de logiciels libres sont généralement des volontaires non payés.

Les F/OSS sont connus pour leur non conformisme par rapport à la rigueur procédurale imposée dans le **développement fermé**[5] de logiciels. Ainsi, les programmeurs dans ces communautés sont libres de choisir les composants sur lesquels ils travailleront en relation à leur expertise. De par le développement ouvert des F/OSS et le non respect des normes de l'industrie dans les projets libres, les considérations de coûts et délais de développement en logiciels libres diffèrent grandement de celles du développement de logiciels propriétaires. En plus du mode de développement libre qui favorise un développement itératif et en parallèle [91] ainsi que la participation[6] de n'importe quel membre de la communauté libre à n'importe quelle phase du processus de développement libre [102], la passion qu'ont les développeurs dans les communautés de logiciels libres leur permet de s'investir et de rendre le code aussi efficace et fiable que possible. Les deux, ensemble (mode de développement libre et passion rencontrée chez les développeurs), permettent de rapidement détecter et corriger les défauts de logiciel. Cela conduit ainsi à une efficacité et une fiabilité accrues du code, mais est également la base de certains problèmes de qualités rencontrés dans les projets. La section 3.2.4 de ce chapitre présente quelques problèmes de qualité rencontrés identifiés dans les F/OSS.

---

[5] Utilisé pour référencer le développement de logiciels propriétaires. Les entreprises commerciales suivent habituellement des normes de génie logiciel afin d'assurer et de maintenir la qualité de leur processus, ainsi que celle de leurs produits logiciels.

[6] Dans les communautés de F/OSS, tant des individus volontaires participent aux projets, mais également des entreprises et leurs employées [110].

3.1.3 Principes derrière le libre : cas de quelques projets

Chaque participant d'un projet de F/OSS respecte les principes soutenant le développement dans sa communauté. Bien que ces principes diffèrent d'un projet à un autre, nous avons cependant noté des similitudes entre les communautés grâce aux nombreuses littératures faites dans les F/OSS. Ces similitudes consistent notamment: aux quatre libertés du libre, traiter les utilisateurs comme des co-développeurs, à rapidement distribuer le code, à livrer rapidement le logiciel et à utiliser des outils libres ou open source afin de soutenir le développement.

Afin de regrouper ou généraliser les F/OSS, nous avions pensé identifier les principes globaux des logiciels libres. Cependant, nous n'avons retrouvé que très peu d'informations à ce sujet. Très peu d'auteurs et de F/OSS ont identifié les principes soutenant le développement libre. Ainsi dans cette section, nous présentons uniquement les principes de quelques projets populaires de F/OSS.

3.1.3.1 Linux

Son nom initial est GNU/Linux faisant référence au projet GNU [101]. GNU/Linux est un système d'exploitation libre lancé en 1983 par Stallam et maintenu dans le projet GNU [101], et plus connu sous le terme Linux. Ce système utilise le noyau libre de système d'exploitation crée par Linus Dorvals en 1991.

Les principes soutenant Linux tel qu'identifiés par Raymond [102], bien qu'abstraits, sont les suivants :

1. *Tout bon logiciel commence par gratter un développeur là où ça le démange.*
2. *Les bons programmeurs savent quoi écrire. Les grands programmeurs savent quoi réécrire (et réutiliser).*

3. *Si vous avez la bonne attitude, les problèmes intéressants viendront à vous.*
4. *Quand un programme ne vous intéresse plus, votre dernier devoir à son égard est de le confier à un successeur compétent.*
5. *Traiter vos utilisateurs en tant que co-développeurs est le chemin le moins semé d'embûches vers une amélioration rapide du code et un débogage efficace.*
6. *Distribuez tôt. Mettez à jour souvent. Et soyez à l'écoute de vos clients.*
7. *Étant donné un ensemble de bêta-testeurs et de co-développeurs suffisamment grand, chaque problème sera rapidement isolé, et sa solution semblera évidente à quelqu'un.*
8. *Il vaut mieux avoir des structures de données intelligentes et un code stupide que le contraire.*
9. *Si vous traitez vos bêta-testeurs comme ce que vous avez de plus cher au monde, ils réagiront en devenant effectivement ce que vous avez de plus cher au monde.*
10. *Il est presque aussi important de savoir reconnaître les bonnes idées de vos utilisateurs que d'avoir de bonnes idées vous-même. C'est même préférable, parfois.*
11. *Bien souvent, les solutions les plus innovantes, les plus frappantes, apparaissent lorsque vous réalisez que votre approche du problème était mauvaise.*

3.1.3.2 Mozilla

Fondé en 1998, Mozilla est un projet *open source* qui se concentre à la promotion de l'ouverture, de l'innovation et des possibilités que le web peut offrir [2]. Mozilla est une organisation hybride qui combine à la fois le but non lucratif et les stratégies de marché pour assurer que l'Internet demeure une ressource publique et partagée [14]. En voici un exemple d'application Mozilla : le navigateur Firefox. Firefox est conçu

pour différentes plateformes : les ordinateurs (les portables et les ordinateurs de bureau) et le mobile (particulièrement le système Android) [95].

Les principes derrière Mozilla, énoncés dans le Manifeste de Mozilla [22] sont les suivants :

1. *Internet fait partie intégrante de la vie moderne — il s'agit d'un composant clé dans l'éducation, la communication, la collaboration, les affaires, le divertissement et la société en général.*
2. *Internet est une ressource publique mondiale qui doit demeurer ouverte et accessible.*
3. *Internet doit enrichir la vie de tout le monde.*
4. *La sécurité des personnes sur Internet est fondamentale et ne peut pas être considérée comme optionnelle.*
5. *Chacun doit avoir la possibilité de façonner son utilisation d'Internet.*
6. *La réalité d'Internet en tant que ressource publique dépend de l'interopérabilité (des protocoles, des formats de données, du contenu), de l'innovation et d'une participation décentralisée mondiale.*
7. *Les logiciels libres et open source favorisent le développement d'Internet comme ressource publique.*
8. *Des processus transparents et communautaires favorisent la participation, la responsabilité et la confiance.*
9. *L'investissement commercial dans le développement d'Internet apporte de nombreux bénéfices ; un équilibre entre les objectifs commerciaux et l'intérêt public est crucial.*
10. *L'amplification des aspects d'intérêt public d'Internet est un but important, digne du temps, de l'attention et de l'investissement que l'on y consacre.*

3.1.3.3 Apache

Le projet Apache est un projet ayant vu le jour en 1995, et est constitué d'une fondation (ASF : Apache Software Foundation) [9]. ASF est gérée par deux principales entités :

- **Le conseil d'administration** : Il dirige la fondation Apache et est composé couramment de neuf individus élus chaque année parmi les membres de l'ASF [9]. Le conseil est responsable de la gestion et de la supervision des activités et des affaires de la société conformément aux statuts de la fondation Apache. Plus en détails, le conseil assure la gestion des actifs (fonds, la propriété intellectuelle, des marques de commerce et l'équipement de soutien) et l'allocation des ressources de l'entreprise à des projets.
- **Les comités de gestion de projets (PMC)** : Ils dirigent le projet et sont constitués des *committers*[7]. Chaque PMC pour un projet donné dans Apache, constitue l'autorité de prise de décision technique concernant le contenu et la direction de ce projet dans Apache. Plus en détails, le rôle du PMC se divise en deux volets, le premier étant de : *s'assurer que toutes les questions juridiques sont abordées, s'assurer que la procédure soit suivie et que chaque version du logiciel soit le produit de la communauté dans son ensemble.* Le deuxième volet consiste à : *promouvoir le développement à long terme et la santé de la communauté dans son ensemble, de faire en sorte que la revue par les pairs soit équilibrée et à large échelle, et que la collaboration entre les pairs se produise.*

---

[7] Représente un développeur ayant le droit d'accéder et d'écrire au dépôt de code, et ayant signé un Contrat de Licence de Contributeur (en anglais *Contributor License Agreement* (CLA)) [9]. Le CLA est une licence dans Apache protégeant la propriété intellectuelle ayant été contribuée à la ASF [6].

Les 6 principes cités comme croyances fondamentales de la philosophie derrière la fondation Apache (ASF) [9] sont :

1. *Développement de logiciels de collaboration*
2. *Licence standard librement commerciale*
3. *Logiciels constamment de haute qualité*
4. *Interaction basée sur la technique, l'honnêteté, le respect*
5. *Mise en œuvre fidèle des normes*
6. *Fonctionnalité obligatoire telle que la sécurité*

Comme exemple de projets Apache, nous citons les suivants : Apache http server, Ofbiz, Anika [30].

3.1.4 Style de développement

3.1.4.1 Type de développement

Le mode de développement utilisé dans les F/OSS tend à rejoindre le développement de logiciel agile [38]. Les communautés de F/OSS utilisent un modèle *hybride* de développement qui est un développement évolutif, asynchrone et ouvert [92], à la différence du développement commercial qui est fermé. Cette idée est soutenue par Jørgensen qui qualifie le développement de logiciel libre de développement incrémental [96].

- ✓ **Ouvert :** il peut s'expliquer par le fait que, le code est disponible pour tous les participants au projet. De même, l'ouverture fait référence au fait que tous les processus de développement, processus de changements, les fonctionnalités et les améliorations, sont visibles sur portail de chaque projet et sont à la portée de tous.

- ✓ **Asynchrone :** Otto dans le dictionnaire informatique [99], définie ce terme de la façon suivante :

« *Asynchrone, désigne un type d'échange de données entre deux machines où les données échangées sont émises et analysées selon une référence de temps différente et un rythme variable. Désigne également le même modèle de communication que précédemment mais cette fois appliqué à des processus.* »

L'aspect important à retenir dans cette définition pour le contexte de logiciel libre est qu'il est appliqué aux différents processus de développement. Le fait que les programmeurs en F/OSS, étant principalement des bénévoles, soient géographiquement distribués à travers le monde et qu'ils communiquent par l'Internet, conduit à des rythmes de travail variables. Par exemple, prenons le cas d'une revue de code par les pairs dans les F/OSS. Habituellement, chaque revue se fait sur une période de temps définie. Tout participant à cette revue travaille indépendamment sur sa copie local du code, et à son rythme. Les corrections de code sont faites parallèlement.

- ✓ **Évolutif :** L'évolution d'un point de vue programmeurs ou utilisateurs de logiciels libres, peut se voir comme « itératif + incrémental ». L'idée dans « itératif » est de fournir le plus tôt possible et de manière fréquente une version du système qui marche, avec des fonctionnalités basiques et très souvent une partie des fonctionnalités ayant été soigneusement intégrées et testées. Chaque itération contient les phases suivantes : étude de la faisabilité, l'élaboration, l'implémentation ou fabrication du logiciel et pour finir la livraison. À chaque nouvelle itération, de nouveaux besoins sont examinés. L'idée dans « incrémental » en logiciel libre, consiste à rajouter des fonctionnalités et informations supplémentaires à la version précédente du système, développée lors du cycle de

développement précédent. L'évolutivité met donc en œuvre l'aspect cyclique du processus de développement de F/OSS.

En 1999, les enquêtes faites par Raymond dans son livre intitulé « *The Cathedral and the Bazaar* » [102], nous montrent que le modèle de développement des logiciels libres, ressemble plus à un « bazar », ceci à l'opposé du modèle « cathédrale » qui est celui des logiciels propriétaires. Le style bazar des F/OSS selon Raymond est caractérisé par la nature non-organisée et très ouverte dans laquelle tout le monde peut participer. Raymond résume ainsi le développement de logiciel libre par la phrase « ***distribuez tôt, distribuez souvent*** » [102]. Également, Raymond mentionne que cette approche de développement encourage activement les utilisateurs à participer aux projets et à contribuer de diverses manières à savoir : *soumission des rapports de bogues, procéder aux revues de code ou suggérer de nouvelles fonctionnalités*. De plus, ce mode de développement compte sur un prototypage rapide dont le développement est fait de manière itérative. Ce mode de développement a été initié par Linus Dorval dont la règle d'or est de : « *distribuez vite et souvent, déléguez tout ce que vous pouvez déléguer, soyez ouvert jusqu'à la promiscuité* ». Dans cette façon de développer, le travail est fait en parallèle. Par exemple au même moment que se fait le développement, particulièrement, l'élimination des défauts est également faite [91]. Ce processus de développement en parallèle procure ainsi un niveau élevé de qualité des produits logiciels développés [91]. Raymond dans le même ouvrage, a eu à identifier quelques facteurs contribuant au niveau élevé de qualité dans les projets de F/OSS, notamment :

- **Les revues par les pairs :** tant les développeurs que les utilisateurs peuvent réviser et modifier le code source, car les projets libres sont par définition ouverts et transparents;
- **La motivation des participants :** les participants dans les F/OSS étant principalement bénévoles, ils sont motivés par une passion pour le travail

qu'ils réalisent [80]. Ainsi, particulièrement pour des programmeurs, cette passion mène à de la créativité et à beaucoup d'innovation. Du côté des utilisateurs, le cycle de développement rapide des F/OSS permet d'obtenir des feedbacks rapides des utilisateurs, lesquels favorisent grandement la qualité;

- **Très peu de contraintes :** le manque de contraintes liées à la rigueur procédurale, comme tel est le cas pour un développement en entreprise, contribue également à la qualité des F/OSS car, cela permet de trouver de meilleures solutions.
- **Des meilleures personnes :** les F/OSS attirent des personnes très qualifiées qui sont des experts en leurs domaines.

Des années plus tard, précisément en 2007, Kevin Crowston et ses collaborateurs [38] présentent un nouveau modèle de développement qui se concentre particulièrement sur la structure sociale des F/OSS. Ce modèle porte le nom du modèle en oignon [26, 38]. Il fait participer les utilisateurs de logiciels aux projets de F/OSS [31] pour que le logiciel respecte le plus possible leurs exigences. Kevin Crowston et ses collaborateurs [38], notent que les F/OSS utilisent un modèle de développement se rapprochant du modèle agile, qui vise à impliquer le plus possible les clients ou les utilisateurs et qui permet aux développeurs de réagir rapidement à leurs demandes. Selon ce modèle, une communauté durable de F/OSS est constituée d'un petit groupe de membres de base, d'un nombre croissant de développeurs contribuant au projet et un nombre élevé d'utilisateurs actifs signalant des défauts [26].

La figure 3.1 représente la structure de l'équipe de développement dans les projets de F/OSS, basée sur le modèle en oignon [38]. De manière hypothétique comme le mentionnent Kevin Crowston et ses collaborateurs [38], parmi les développeurs appartenant au groupe noyau, ici, des développeurs de confiance ayant certains

privilèges tel que l'accès au référentiel de contrôle de version, il y a l'initiateur du projet, le coordonnateur de versions.

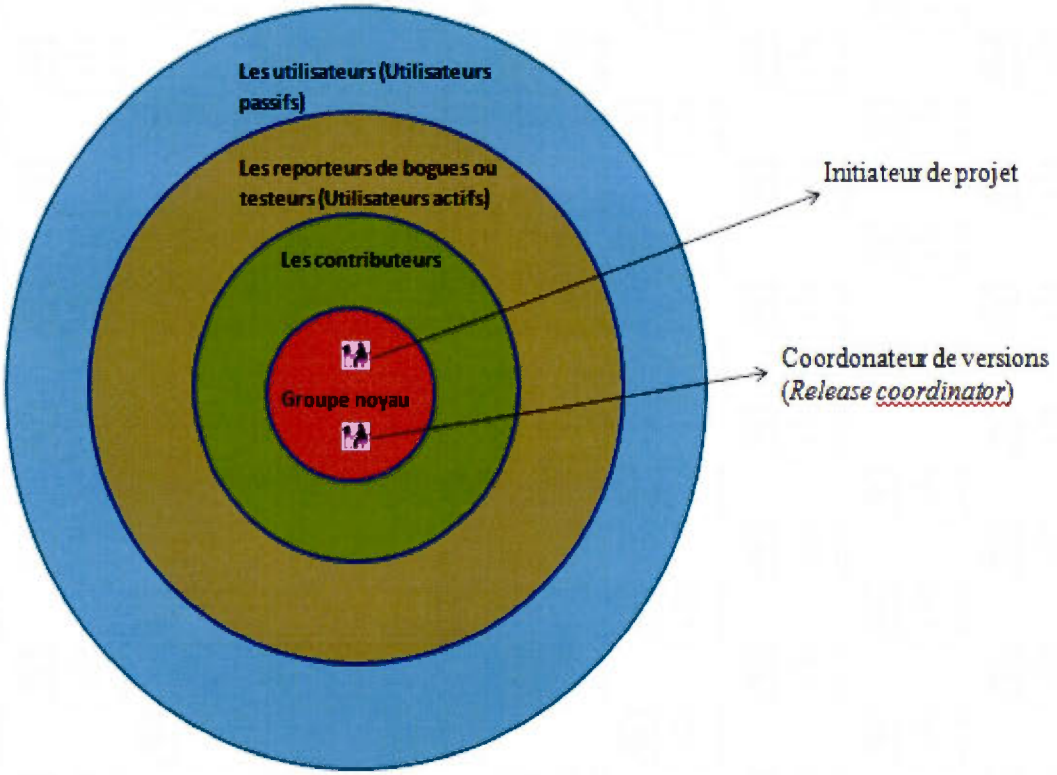

Figure 3.1 Structure de l'équipe de développement des F/OSS

Dans le modèle en oignon, les participants actifs au projet peuvent monter en grade au niveau de la prochaine couche fermée la plus interne, en gagnant le respect de la communauté à travers un processus de méritocratie[8] [26, 38]. Tout comme l'a

[8] C'est une philosophie fonctionnant comme suit: plus des participants aux projets de F/OSS fournissent des contributions signifiantes, plus ils gagnent des responsabilités au sein de la

mentionné Raymond [102], les utilisateurs peuvent participer aux projets en envoyant des rapports de bogues, des suggestions ou par d'autres feedbacks. C'est ainsi qu'ils deviennent des membres actifs de la communauté. À ce stade, chaque participant peut devenir un développeur, contributeur au projet s'il a soumis bon nombre de contributions importantes. Quant à la couche la plus interne, ici, groupe noyau, elle est réservée aux membres actifs et ayant le plus d'expérience au sein de la communauté. Ces membres ont certains privilèges, comme l'accès au référentiel de documents. Chaque rôle dans le modèle est associé à un nombre de tâches et à des responsabilités spécifiques :

- **Reporteur de bogues ou testeurs :** L'un des points forts des F/OSS est qu'un très grand nombre de participants au projet, particulièrement des utilisateurs, fournissent des feedbacks usuels [102]. Ces feedbacks, peuvent être sous forme de rapports de bogues, de suggestions de nouvelles fonctionnalités et d'exigences [19, 91, 102]. La majorité des défauts majeurs du logiciel sont détectés par les utilisateurs.
- **Contributeurs :** Ce sont des développeurs appartenant à un groupe fermé, contribuant par de nouvelles fonctionnalités pour l'intérêt du projet [91].
- **Équipe de base ou groupe noyau :** Ce sont des contributeurs ayant fournis bon nombre de contributions importantes, se voyant ainsi gagner le respect des autres membres de la communauté de F/OSS. Ils ont pour rôles selon Lerner et Triole [83]: de fournir la vision du projet, de diviser le projet en plusieurs parties dans lesquelles les individus peuvent s'attaquer aux tâches indépendantes, d'attirer les développeurs au projet et de garder le projet ensemble et de prévenir la bifurcation.

---

communauté. Pour de nouveaux adhérant au projet, ils gagnent un accès direct au dépôt de code. Et s'ils sont très actifs, ils peuvent grimper en échelon.

3.2 LITTÉRATURES SUR LA QUALITÉ DANS LES F/OSS

3.2.1 Définition de la qualité

Le concept de qualité de logiciel est un concept très intéressant. Cependant, quand vient le moment de définir la qualité de manière précise, ce concept devient un peu plus ambigu, car il nécessite une bonne compréhension du processus de prise de décision menant à la qualité. Bon nombre de personnes sont à même de reconnaitre la qualité d'un logiciel quant elles la voient, cependant, elles sont incapables de la définir de façon précise, encore moins de décrire le processus menant à cette qualité. Face à tout ce désordre et au besoin d'avoir une définition et une compréhension universelle de la qualité, le sous-groupe CEI (Commission Électronique Internationale), de l'organisme ISO a proposé la norme ISO/CEI 9126 [71].

La qualité au sens de la norme ISO/CEI 9126 [71] est un modèle de qualité constitué d'un ensemble **d'attributs de qualité** (qualité interne, qualité externe et qualité fonctionnelle de produit logiciel) que doit respecter un produit logiciel. Cette approche de qualité est une perspective orientée utilisateur. Les attributs de qualité identifiés dans ISO/CEI 9126 sont regroupés dans les six caractéristiques suivantes:

- **La fonctionabilité :** c'est la capacité du produit logiciel à fournir des fonctions qui répondent à des besoins exprimés et implicites lorsque le logiciel est utilisé dans des conditions spécifiées;
- **La fiabilité :** c'est la capacité du produit logiciel à maintenir un niveau de service spécifié lorsqu'il est utilisé dans des conditions précises;
- **L'utilisabilité :** capacité du produit logiciel à être compris, connu, utilisé et à plaire à l'utilisateur dans des conditions spécifiées d'utilisation;

- **L'efficacité :** capacité du produit logiciel à fournir des performances appropriées en fonction de la quantité de ressources utilisées, dans des conditions données;
- **La maintenabilité :** capacité du produit logiciel à être modifié;
- **La portabilité :** capacité du produit logiciel à être transféré d'un environnement à un autre.

Cette approche permet non seulement de déterminer si un produit logiciel est apte à l'emploi conformément aux exigences définies, mais également de déterminer ses performances (fiabilité, efficacité, portabilité, utilisabilité et maintenabilité), afin de déterminer son niveau de qualité.

Quant à l'application de cette approche de qualité dans les F/OSS, de nombreuses questions sont soulevées, notamment les différents facteurs à considérer pour atteindre un certain niveau de qualité du logiciel. Les F/OSS sont connus pour abriter essentiellement des membres bénévoles. Ce contexte est donc conflictuel avec l'approche de la norme ISO/CEI 9126, car les développeurs des F/OSS comme cela est le cas dans les entreprises commerciales, peuvent décider de ne mettre l'emphase que sur certains attributs de qualité prescrit par ISO/CEI 9126.

La vision de qualité diffère du point de vue des programmeurs par rapport aux utilisateurs. Le plus souvent, la différence entre les programmeurs et les utilisateurs se résume au fait que les utilisateurs se concentrent particulièrement sur les attributs de qualité suivants : utilisabilité, fiabilité et fonctionalibité; tandis que les programmeurs se concentrent sur la portabilité, la maintenabilité et l'efficacité, soutenus par l'attribut de qualité fonctionabilité. Les utilisateurs exigent des mesures et des outils pour mesurer la qualité d'un logiciel [115]. En logiciel libre, tel qu'identifiés par Tawileh et Rana [115], les développeurs croient fermement que les

idées derrière le modèle de développement de F/OSS, en l'occurrence, les idées de communication intensive entre les développeurs, la large dépendance dans les revues par les pairs et une livraison fréquente du code source [102], sont suffisantes pour atteindre un niveau de qualité et de fiabilité élevées dans les produits logiciels développés. Cela étant, ces idées permettent : d'accélérer la détection et correction des bogues, de fixer la date de livraison et d'améliorer la qualité des produits logiciels. Certains projets F/OSS ont opté pour des outils de type F/OSS leur permettant d'évaluer la qualité du code de leur produits logiciels en examinant certains aspects. Par exemple, SourceForge héberge PMD [112] qui est un analyseur de code Java, permettant de retrouver certains bogues, tel que les blocks de catch vides, la création d'objets inutile; de retrouver des variables non utilisées, de retrouver du code dupliqué. Il en existe plein d'autres outils de ce genre qui soient de type F/OSS. Spinellis et ses collaborateurs [114], ont fait une observation de la qualité de logiciels libre : SQO- OSS (Software Quality Observator pour des produits logiciels OSS) qui se différencie des analyses de la qualité faites par des outils libres d'évaluation de la qualité des F/OSS, car elle permet de calculer et d'intégrer les mesures de divers produits et procédés industriels. Le logiciel produit par la recherche de Spinellis et ses collaborateurs est appelé Alitheia [114]. Il ne se limite pas à un seul langage de programmation comme il est souvent le cas de la plupart d'outils libres d'analyse de code en F/OSS [114].

3.2.2 Comparaison de F/OSS et logiciels commerciaux

Certains logiciels libres sont d'une qualité comparable, voir supérieure aux logiciels propriétaires [67, 108]. L'une des raisons conduisant à cette qualité élevée dans les logiciels libres est que, contrairement au développement traditionnel de logiciels, les développeurs dans les F/OSS sont eux-mêmes les utilisateurs du logiciel développé ou maintenu, et ils sont donc plus pointilleux dans leur développement, rendant ainsi le logiciel aussi efficace et fiable que possible [70, 102]. Cependant, la nature non-

organisée du processus de développement des logiciels libres ainsi que son manque apparent de mécanismes d'assurance qualité [115], contrairement au développement dans les industries, posent un réel problème à ces dernières car elles s'avèrent dépendre de plus en plus des produits développés dans les communautés de F/OSS [91]. Afin de pallier à ce problème qualité des logiciels libres, des modèles de maturité et d'évaluation, particulièrement, l'*Open Source Maturity Model* (OSMM) [63] et le *Business Readiness Rating* (BBR) [1], ont été développés pour évaluer la durabilité d'un projet de F/OSS.

Mockus et ses collaborateurs [94] ont eu à identifier quatre différences entre le développement de F/OSS et celui de logiciels commerciaux. Ces différences sont les suivantes : «

- *Les F/OSS sont construits par un nombre potentiellement élevé de bénévoles;*
- *Le travail n'est pas assigné aux participants, les personnes entreprennent les travaux qu'elles désirent faire*;
- *Il n'y a aucune conception explicite au niveau du système, ou même une conception détaillée du système;*
- *Il n'y a aucun plan de projet, de calendrier, ou liste des livrables.* »

Aberdour [26] quant à lui, a eu à prolonger le travail de Mockus et ses collaborateurs [94], en identifiant des différences en termes de pratiques de gestion de la qualité entre le développement de F/OSS et le développement en entreprise (voir tableau 3.2).

Code source fermé	F/OSS
Méthodologie de développement bien définie	Méthodologie de développement souvent non définie ou documentée
Tests et méthodologie d'assurance qualité structurés et formels	Tests et méthodologie d'assurance qualité non structurés et informels
Les analystes définissent les exigences	Les programmeurs définissent les exigences
Objectifs mesurables utilisés tout au long du projet	Peu d'objectifs mesurables sont utilisés
Le travail est assigné aux membres de l'équipe	Le travail est choisi par les membres de l'équipe
Phase de conception formelle est effectuée et signée avant que la programmation ne commence	Les projets vont souvent aller directement à la programmation

Tableau 3.2 Comparaison des pratiques de gestion de la qualité [26]

3.2.3 Développement et pratiques de qualité

3.2.3.1 Pratiques de qualité

Les enquêtes menées par Michlmayr, Hunt et Probert [92] en 2005 ont conduit à l'identification de certaines pratiques de qualité dans les F/OSS. Michlmayr, Hunt et Probert [92] ont eu à les regrouper en trois principaux groupes: **infrastructure**, **processus** et **documentation**.

- **Infrastructure :** les F/OSS sont constitués d'une infrastructure solide qui est la base de tout développement mené dans ces communautés. Cette infrastructure est essentiellement constituée des : systèmes de gestion de

version, système de suivis de bogues, les « *builds* » automatiques et des listes de diffusion (voir section 3.2.3.2.3 pour les détails). Bien que de nombreux F/OSS utilisent une telle infrastructure, la façon dont ils l'utilisent diffère grandement d'un F/OSS à un autre [91]. La manière d'utiliser cette infrastructure peut contribuer à une qualité élevée [91]. Par exemple, certaines communautés utilisent des « builds » automatiques (par exemple, Mozilla [94, 104]) et tant qu'un défaut n'a pas été supprimé, le logiciel ne passe pas au prochain test;

- **Processus :** les processus de F/OSS identifiés par Michlmayr [91, 92] sont :
    - **L'adhésion :** certaines communautés de F/OSS utilisent à ce niveau comme pratique de qualité, d'accepter uniquement des contributeurs ayant déjà eu à soumettre des contributions de qualité élevée [91]. D'autres projets libres, quant à eux sont plus libéraux dans leur choix.
    - **La release :** au niveau de la « release », les phases de gel du code (*freeze stages*) sont appliquées par plusieurs F/OSS [60, 91], conformément à leur politique respective de release. Le gel du code est une fonctionnalité utilisée dans les F/OSS afin de ne permettre aucune incorporation de nouvelle fonctionnalité dans le code de base du logiciel, cependant, ce gel de code sert également de période pour corriger les défauts du logiciel.
    - **Les branches :** elles sont très répandues en logiciels libres [28, 60]. Elles sont utilisées pour la gestion des versions de programmes. Certains projets libres utilisent des outils afin de faciliter les branches dans les programmes. L'usage de tels outils fait toute la différence dans leur processus de développement [91].
    - **Les revues par les pairs :** de nombreux auteurs [26, 36, 38, 52, 60, 91, 92, 105, 115] sont arrivés à la conclusion que les revues par les

pairs contribuent grandement à la qualité des logiciels développés ou maintenus en logiciels libres. Cependant, ce processus de revues par les pairs diffère d'un projet à un autre. Par exemple, certains projets utilisent un processus rigoureux de revues du code avant de le soumettre au référentiel de documents du projet et d'autres quant à eux, font une revue pas très formalisée.

- **Les tests :** Michlmayr [91] a eu à identifier que certains projets de F/OSS appliquent des listes de contrôle de tests afin de s'assurer que la nouvelle version du logiciel répond aux normes du projet et qu'elle ne contient pas des régressions majeures.
- **L'assurance qualité :** particulièrement dans le cadre de la gestion des versions, Michlmayr [91] a eu à identifier que certains F/OSS organisent des journées de bogues, consistant à trier les bogues encore non résolus. Les revues récurrentes par les pairs sont réalisées dans les F/OSS [52, 102, 115] et elles permettent la participation de la communauté toute entière de chaque projet de F/OSS [102]. Cela permet que le logiciel puisse atteindre un niveau élevé de qualité et qu'il puisse respecter les besoins des utilisateurs.

- **Documentation :** les F/OSS souffrent d'un manque accru de documentation dans leur processus de développement [92]. Quelques fois, des codes sources contrôlés par un développeur chef peuvent être rejetés faute d'un non respect du style de codage, car ce style n'est pas explicitement documenté quelque part sur le portail web de la communauté. Face à cela, des projets à grand nombre de contributeurs ont développé de bonnes documentations, en l'occurrence, les documentations de *style de codage*[9] et les documentations de

[9] C'est une documentation destinée aux développeurs qui décrit le style qui doit être utilisé pour le

soumission de code[10] [91], qui sont des conventions propres à chaque communauté libre. Ces dernières sont considérées comme des pratiques de qualité.

3.2.3.2 Quelques éléments de base du développement des F/OSS

De façon globale, les F/OSS fonctionnent tels que présentés dans la figure 3.2.

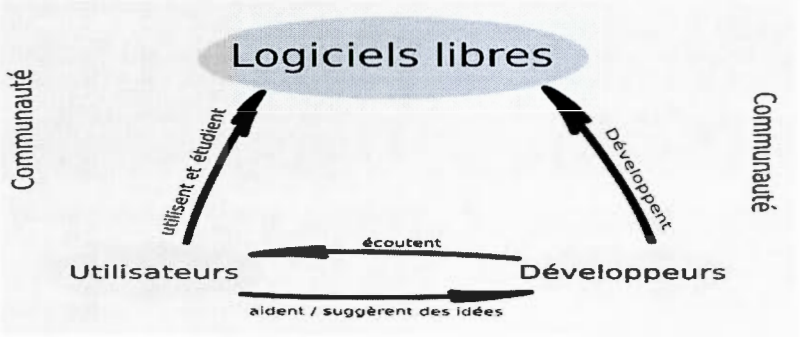

Figure 3.2 Fonctionnement des communautés de logiciels libres [117]

3.2.3.2.1 Adhésion, initiation et planification de projet

La *gestion de projet* en F/OSS, telle que définit par Mockus et ses collaborateurs [94] est : « l'initiation et la planification d'un projet, de manière à attirer le maximum de personnes ». La gestion de projet est l'un des problèmes que rencontrent les communautés de logiciels libres depuis des années. Ce problème provient du fait que les projets de logiciels libres sont tenus par des participants volontaires, non payés, répartis à travers le monde [52, 94]. Une description détaillée du processus d'adhésion aux F/OSS a été faite en 2003 par Krogh et ses collaborateurs [58].

---

code source [91].

[10] C'est la documentation qui décrit quand et qui peut apporter des changements dans le système d'un projet de contrôle de version [91].

3.2.3.2.2 Implémentation de logiciel

Le plus souvent dans des projets de F/OSS, il n'existe pas de phases explicites de conception de logiciel, ni de conception détaillée [35, 57, 79, 94, 104]. En 1999, il y a eu des divergences d'opinions sur la qualité des logiciels développés dans les communautés de F/OSS. Ainsi, certains auteurs à l'instar de McConnell [90] trouvent que : les bogues sont corrigés au niveau du code au lieu du niveau de conception dans le développement de F/OSS. Par conséquent, cela irait à l'encontre des pratiques reconnues dans l'industrie, et donc, cela pourrait faire sombrer certains grands projets. D'autres chercheurs à l'instar de Bollinger et ses collaborateurs [36] trouvent que les défauts sont détectés bien avant le niveau du code, car les revues par les pairs sont faites très tôt, fréquemment et de manière itérative; permettant ainsi d'éviter d'avoir des bogues dans les produits logiciels en maturité[11]. L'idée de Bollinger et ses collaborateurs [36] est soutenue par Rigby, German et Storey dans le cadre du projet Apache [105]. Au cours des études menées sur le développement Open Source, Mark Aberdour [26] affirme qu'il n'existe pas de processus rigoureux de test de système dans le développement F/OSS contrairement au développement à code source fermé :

« *Dans le développement Open Source, le succès de la détection et correction des erreurs à travers le cycle de vie et particulièrement pendant les tests du système, dépend moins de la rigueur procédurale. Au lieu, il emploie une variété de techniques pour aboutir à une qualité élevée.* » [26]

Les techniques dont parle Mark Aberdour [26] consistent en :

- Des revues qui se font tôt et de manière régulière. Cela permet aux

[11] La maturité du produit logiciel à ce niveau s'appréhende à une version stable du logiciel, telle est la conception de Bollinger et ses collaborateurs [36].

développeurs de détecter et de corriger des erreurs le plus tôt possible et de manière continue et itérative, jusqu'à ce qu'il n'y ait plus d'erreurs [102, 105, 115];

- Le nombre important de personnes participant aux revues, permet de facilement détecter des bogues [52, 94].

Certaines enquêtes dans les F/OSS ont identifié la modularité du code comme facteur de qualité dans le développement de F/OSS [26, 36, 60, 104, 105] car elle:

- Limite les risques de propagation des bogues entre les modules de code;
- Permet d'avoir des codes très efficaces : le fait que plusieurs développeurs peuvent travailler sur les mêmes solutions, en laissant parler leur créativité, accroit la probabilité de trouver des solutions de qualité élevée en un court laps de temps;
- Facilite la correction des bogues car les modules sont moins complexes et facilement compréhensibles.

Allant dans le même sens, Reis et Fortes [104], mentionnent que la grande modularité du code:

- Permet la réutilisabilité des composants logiciels;
- Permet aux développeurs de se concentrer sur des fragments de code sans nécessité de comprendre l'ensemble de l'architecture logicielle;
- Permet un apprentissage progressif, particulièrement pour les nouveaux participants au projet;
- Rend possible le développement du logiciel, ceci en dépit de la concurrence naturelle présentée par une communauté de développeurs répartis.

Certains projets de F/OSS peuvent ne pas continuer faute d'un nombre réduit de développeurs. Juan C. et ses collaborateurs [44] identifient que, pour qu'un projet dans le monde du libre ait des chances d'émerger, le nombre moyen de participants développeurs doit être **au moins** de 15. Peter C. et ses collaborateurs [105] ainsi que Aberdour [26], ont trouvé des résultats semblables dans la communauté d'Apache. Dans le même contexte, certaines recherches menées sur les F/OSS ont aboutis aux résultats suivants : Mozilla est constitué de 22 développeurs de confiance [94], quant à GNOME, il en est constitué de 52 [78]. D'autre part, des recherches rapportées par Michlmayr et ses collaborateurs [92] ont montré que les erreurs dans les projets peuvent être liées au manque de compétences en gestion de projets.

Exemple:

➢ **Cas d'Eclipse :**

Dans *Eclipse Fundation* [47], chaque projet possède une équipe de développement[12], lequel projet est conduit par une direction de projet (*Projects leardership*), composée des :

- **Contributeurs** : ce sont des développeurs qui contribuent au code, aux tests, à la documentation, etc…
- ***Committers*** : ce sont des développeurs qui définissent et contrôlent l'accès au *référentiel de contrôle de versions du code source* (en anglais *repository*) du projet et influencent le développement du projet. Ils sont élus par vote pendant une période d'au moins une semaine, à travers un processus d'élection transparent et ouvert. Un *committer* a pour tâches spécifiques de : participer activement aux nouveaux groupes d'utilisateurs et à voir les listes appropriées des listes de diffusion des développeurs (*mailing lists*). Ces tâches sont critiques pour le succès d'un projet. En gros, les *committers* :

[12] Les différents rôles dans le projet Eclipse sont : *Committer*, chef du PMC, Chef de projet, représentant du conseil. Un même individu peut porter plusieurs chapeaux parmi ces rôles [47].

- surveillent et contribuent aux nouveaux groupes utilisateurs et aux *mailing lists*, associés à un projet;
- suivent, participent et portent des votes sur des discussions importantes dans leurs projets et composants associés. Trois réponses de vote existent: +1 (oui), -1 (non, ou droit de veto), et 0 (s'abstenir);
- sont responsables de façon proactive des problèmes dans le système de suivi des bogues. Ils annotent les rapports de problèmes avec les informations sur l'état, des explications, des clarifications ou des demandes d'informations additionnelles de l'expéditeur. De plus, ils sont responsables de la mise à jour des rapports de problèmes quand ils ont fait des travaux liés à ce problème.

3.2.3.2.3 Gestion de configurations, de documents et de versions

Étant donné le volontarisme, le bénévolat et la distribution géographique des participants aux projets de logiciels libres et en considérant le mode de développement des F/OSS, l'Internet reste l'unique moyen de communication et est nécessaire à la bonne marche des F/OSS. Cela a pour conséquence implicite que toute l'information soit électronique et gérée par le biais de l'infrastructure en place. Et donc, l'omniprésence de l'Internet dans les F/OSS soulève un réel problème dans la gestion des documents et du matériel, liée aux projets de F/OSS et même un problème au niveau de la gestion du personnel. Comme le mentionnent Zu Xiao et ses collaborateurs [122], les communautés de F/OSS n'ont pas de groupe prédéfini de gestion de configuration, à l'opposé du modèle de développement de logiciels commerciaux. Généralement lorsqu'une question portant sur un F/OSS est soulevée, il devrait automatiquement y avoir une réponse à cette question [105]. Pour le bon déroulement des projets de F/OSS, il est plus que nécessaire que les communautés libres aient de bonnes infrastructures afin de pouvoir gérer les projets développés ou maintenus. Cette infrastructure (voir tableau 3.3) permet non seulement de facilement

gérer et maintenir un projet, mais également de construire toute une communauté autour du projet, faisant ainsi participer les utilisateurs.

Pour la gestion de documents par voie électronique, les communautés de F/OSS se doivent d'utiliser des formats standards de documents. Habituellement, le PDF est utilisé comme standard pour des documents.

Projets de F/OSS	**Infrastructure**					
	Système de gestion de versions	*Builds* et tests automatiques	Listes de diffusion Utilisées : **oui /non**	Systèmes de suivis des bogues	Systemes/API de partage de contenus et messagerie instantanée	Système de gestion de changements et visionneurs de code
Eclipse	CVS	Information non trouvée	Oui : Mailman [85]	Bugzilla[47, 67]	Flick, Photobucket, Twitter, SlideShare, facebook [47]	CCB [118]
Apache (HTTP Server)	SVN [30], CVS [67]	Aucun outil de build et de test automatique	Oui : Ezmlm [67]	GNATS [67], BUGDB [94], Bugzilla [25], JIRA [3]	IRC [3]	ViewCVS [67]
Gnome	CVS [39]. Actuellement, ils utilisent Git [41]	Autoconf, automake, vi, gee, boîte à outils gcc[60], et donc,	Oui : Mailman[60, 67]	Bugzilla[67], Debbugs[64]	IRC [67], le site web de GNOME [60]	LXR, bonsai [67]

		implicitement utilise DejaGnu[67]				
Mozilla	CVS	Tinderbox[67], Mozmil[5]	Oui : mailman[86]	Bugzilla[67]	IRC [67]	CCB [118], LXR ou bonsai [67]
Linux	Aucun outil de gestion de version n'est utilisé. Néanmoins, chaque version est placée dans un répertoire différent [118]. CVS et CVSWeb [67], bitkeeper[60]	DejaGNU[67]	Oui : Mailman ou Majordomo[67]	Bugzilla[67], GNATS [67]	Sendmail [67]	LXR [67]

Tableau 3.3 Récapitulatif des outils de gestion de documents et de configurations dans certains projets de F/OSS

3.2.3.2.4 Méthodes de vérification et validation

Les méthodes de vérification et de validation (V&V) sont des techniques du génie logiciel, utilisées en industrie pour pouvoir détecter et corriger le plus tôt possible les défauts/erreurs pouvant survenir dans un logiciel. En d'autres termes, ces méthodes de V&V sont des techniques permettant de déterminer si oui ou non le produit lors

d'une activité du cycle de vie du logiciel satisfait aux exigences définies. Ces techniques peuvent être utilisées individuellement ou de manière jumelées pour plus d'efficacité. Selon la norme IEEE 1059-93 [69], la vérification et validation de logiciel consiste en « *une approche disciplinée de l'évaluation du produit logiciel pendant toute la durée du cycle de vie du produit. Une V&V vise à assurer que la qualité soit intégrée dans le logiciel et que le logiciel répond aux besoins des utilisateurs* ».

Parmi les techniques de V&V, l'on rencontre les techniques suivantes le guide SWEBOK [27]:

- **L'inspection :** selon la norme IEEE 1028-97 [68], l'objectif de l'inspection est de détecter et d'identifier les erreurs du produit logiciel. De par cet objectif, nous dirons ainsi que l'inspection va permettre de détecter et de corriger les erreurs le plus tôt possible dans le cycle de vie de logiciel, de vérifier que le logiciel respecte les exigences du client.
- **Le walkthrought :** selon la norme IEEE 1028-97 [68], l'objectif du walktrought est d'évaluer le produit logiciel. Selon SWEBOK [27], le walkthrought est similaire à l'inspection, mais est conduit de manière moins formelle.
- **Les revues par les pairs ou revues techniques:** son but est d'évaluer le produit logiciel afin qu'il rencontre son aptitude à l'usage prévu et d'identifier les écarts par rapport aux exigences et aux normes [68].

Il y a quelques années déjà que le modèle de développement adopté dans les projets de logiciels libres et *Open source,* s'est avéré être une alternative viable par rapport aux autres modèles de développement de logiciel. Outre le fait que les communautés de développement de logiciels libres, développent des pièces de logiciels intéressantes, il faut noter que plusieurs applications de logiciels libres reflètent des

niveaux de qualité comparables, voire supérieurs aux modèles utilisés par des entreprises développant des logiciels propriétaires [67, 108]. Ce niveau de qualité retrouvé dans les projets de logiciels libres, s'explique en partie par la participation des utilisateurs et d'autre part, par un degré élevé de revues par les pairs, comme le suggèrent Raymond [102], Anas Tawileh et ses collaborateurs [115], German [60] et Fielding [52].

Jai Asundi et ses collaborateurs [31], ainsi que d'autres auteurs [26, 36, 38, 52, 60, 92, 105, 115], trouvent que la technique de V&V la plus utilisée dans les communautés de F/OSS, est la *revue par les pairs* et moins les inspections ou les walkthrough que l'on rencontre le plus souvent dans les logiciels commerciaux. De nombreuses recherches sur la qualité des logiciels libres ont montré que la revue par les pairs a un impact considérable sur la qualité des logiciels libres [26, 50, 67, 105, 115].

Quelques exemples de revues par les pairs en F/OSS

- **Cas Apache server :**

Deux types de revues sont à considérer: RTC (*Review-then-commit*) qui est une revue technique, et CTR (*Commit-then-review*) [105]. La RTC est la première forme de revue, utilisée lorsque les membres du groupe noyau ne sont pas sûres de certaines de leurs contributions ou lorsque les contributions sont faites par les développeurs n'appartenant pas au groupe noyau. Les membres du groupe noyau dans Apache sont des développeurs de confiance ayant le privilège de contrôler l'accès au référentiel de gestion de version partagée, qui est un endroit où les contributions ayant été parfaitement révisées sont sauvegardées. Quant à la CTR, elle est utilisée lorsqu'un programmeur du groupe noyau est confiant de sa contribution, i.e contribution qu'il envoie dans le *repository*.

Selon Rigby, German et Storey [105], ces revues sont décrites de la manière suivante :

- Les revues ne sont pas conduites de manière synchrone entre les développeurs [52]. Le processus de revue commence avec un courriel incluant une contribution, envoyé sur la liste de diffusion [105] et ce processus se termine lorsque tous les défauts associés à cette contribution ont été réparés. Pour RTC, chaque contribution doit contenir le mot clé « *PATCH*[13] » dans le message initial, alors que pour CTR, les courriels des sujets à traiter sont émis avec un CVS ou une SVN;
- RTC nécessite 3 personnes/programmeurs pour chaque revue, alors que CTR exige la participation des membres du groupe noyau. Les participants à une revue dans Apache sont en fait des personnes ayant apportées des réponses (différentes de celles des autres participants) à une contribution. Selon Peter C. et ses collaborateurs [105], RTC a généralement plus de participants et de messages échangés lors d'une revue que CTR : « La moyenne pour RTC est de 2 participants et 2 messages, alors que CTR a un individu et 2 messages ».
- Les contributions en revues sont généralement petites, car de cette manière, la gestion des changements et des messages échangés est plus aisée. Comme appui à cela, Peter C. et ses collaborateurs [105] trouvent les résultats suivants :
    - RTC : Le nombre moyen de lignes de code modifiées est de 25 et pour 80% des contributions soumises, le nombre de lignes modifiées est inférieur à 106. En ce qui concerne des documents qui ont subit une RTC et qui ont été acceptés, la moyenne des lignes de codes changées est de 17.

---

[13] Ce sont des *fragments de code* envoyés par courriel entre les développeurs. Un patch peut être ajouté à un logiciel pour y apporter des modifications mineures.

  - CTR : en moyenne, 15 lignes sont modifiées et pour 80% des contributions soumises, le nombre de lignes changées est inférieur à 70.

- Il faut considérer trois durées : la première durée est celle entre le document engagé et la première réponse. Le processus de révision de document débute par le premier courriel contenant la contribution à réviser. La première réponse représente le temps de préparation d'une nouvelle revue. En moyenne, RTC dure 3 heures, et CTR : 1.7 heures. La deuxième durée est celle entre le document engagé et la dernière réponse. Elle en fait la durée des discussions lors de la revue. En moyenne : RTC dure moins de 19 heures et CTR dure 8.9 heures. Pour finir, la troisième durée est le temps mis pour la validation du document. Dans une revue de type RTC, juste 9% des validations mettent plus d'une semaine. Pour CTR, en moyenne, la phase de validation des corrections apportées à un document dure 1.7 jours. Le temps de la revue par induction devient donc la somme de ces trois temps. Et donc en moyenne, une CTR dure 51.4 heures. Et pour une CTR, juste 5% de problèmes ont été trouvés après une semaine, montrant juste le fait que si les changements n'ont pas été révisés immédiatement, ils ne le seront sûrement pas. Une RTC dure donc en moyenne moins de 190 heures, soit donc 7.9 jours. Ces chiffres et pourcentages ont été déterminés par Peter et ses collaborateurs [105].
- La politique d'Apache nécessite que les programmeurs décrivent les changements faits pour chacune des transactions vers le système de contrôle de versions, afin de différencier les versions. Ces descriptions devront contenir les informations telles que le nom de l'auteur de la contribution et les noms des participants à la revue de cette contribution.

Peter C. et ses collaborateurs [105], définissent ainsi les revues dans Apache comme étant: *« (1) des revues qui se font tôt et fréquemment, (2) sur des contributions petites, indépendantes et complètes, (3) revues conduites de manière asynchrone par un petit groupe potentiellement important d'experts auto - sélectionnés (4) menant à une revue technique par les pairs, effective et efficace.* »

Exemple de revue de code :

Fielding [52], définit la revue par les pairs, sous l'appellation de *processus de développement de logiciels* comme étant : « *Un ensemble de procédures par lesquelles les développeurs prennent les décisions sur le projet et coordonnent leurs efforts* ». Fielding mentionne qu'il n'y a ni président, ni gestionnaire, ni CEO dans Apache pour prendre des décisions, et donc, les programmeurs participants au projet procèdent par des votes via des courriels. Chaque programmeur devra envoyer un courriel via une liste de diffusion au sujet d'une question concernant le projet : '+1'pour un vote positif (oui) et '-1'dans le cas contraire (non). Pour une revue de code (des changements dans le code), trois '+1' et aucun vote négatif sont requis, ce qui limite le nombre de participants à la revue. Les revues durent en général 1 à 2 semaines par programmeur et se fait de manière asynchrone. Fielding mentionne également que les programmeurs sont sélectionnés selon leur mérite[14] (i.e des programmeurs ayant fournis des contributions significatives). Par ailleurs, certains auteurs [44, 105] ont également trouvé les mêmes résultats que Fielding. À la fin de la revue, une version stable du produit est obtenue et un rapport de la revue est produit.

- **Cas Eclipse :**

[14] Ce mérite se réfère à la philosophie de méritocratie précédemment mentionnée.

Tel que présenté dans Eclipse [7], le déroulement d'une revue par les pairs est constituée des étapes suivantes :

- Une revue par les pairs dans Eclipse [47] a une durée d'environ une à deux semaines.
- Elle débute au niveau de l'EMO qui poste le matériel de revue sur le site web du projet au début de la période de revue et elle se termine soit par la fin de la période de revue, soit par un appel optionnel de conférence/conférence technologique (exemple : conférence web) ; cela aussi longtemps que la technologie est disponible à tous les membres et n'encourt aucun coûts supplémentaires pour les participants.
- Les participants sont des membres d'Eclipse et les *committers*. Chaque revue possède une EMO, désignée pour les discussions et les canaux de communication de feedback: un nouveau groupe, une liste de diffusion, ou une partie du public d'autres forums.
- Si une élection de *committer* est nécessaire pour une revue (par exemple, pour une revue de migration (voir ANNEXE B)), alors elle est faite pendant la période prévue pour la revue. Ainsi, l'élection et la revue prendront fin en même temps. Cela permet une livraison rapide et de manière efficace du produit qui en résulte.
- Au même moment que l'ouverture de la revue, une date et une heure de la conférence téléphonique sont annoncées. Il est à noter que : *la date d'appel ne doit pas être inférieure au lendemain et pas plus d'une semaine de jours ouvrables standards après le dernier jour de la revue*. Par exemple, si la revue commence mercredi le 4 jusqu'au mardi le 10, l'appel peut avoir déjà été planifié, pour n'importe quel temps, à partir de mercredi le 11, jusqu'à mercredi le 18.
- Au cours de la conférence téléphonique, la direction du projet (ou EMO nommé représentant de projet) fournit un bref résumé des raisons et

justifications de la transition de phase (confer - ANNEXE B faisant référence à la revue de migration), suivie d'une séance de questions-réponses.

- L'EMO (ED) approuve ou omet l'examen (la revue par les pairs) sur la base des observations du public, la portée du projet et les objectifs de la fondation Eclipse tels que définis dans les statuts. L'EMO (ED) annonce le résultat sur le canal de communication de la revue ayant été définie. La revue est déclarée complète lorsque la période prévue pour la revue a expiré.

➢ **Cas GNOME :**

GNOME est un sous-projet libre de la communauté Linux. Il est l'un des grands succès de projets de F/OSS ayant pour principal but de créer une GUI (*Graphical User Interface*) pour les systèmes Unix [60]. Le code noyau du projet est divisé en modules, pour faciliter le développement de ce projet. Chaque module possède ses propres mainteneurs, un ensemble de développeurs, un calendrier de développement et ses objectifs. Les mainteneurs jouent le rôle de chefs pour le module auquel ils appartiennent. Ils sont des contributeurs de la communauté de GNOME qui ont été élus par les membres de GNOME. Les développeurs sont des contributeurs ayant chacun leur propre temps de soumission de leurs contributions. GNOME possède une équipe de révision pour chacun des modules, dont la responsabilité est *de planifier et de coordonner l'ensemble du projet, de manière à ce que le projet suive le calendrier des révisions (revue par les pairs) proposées*. Les revues de documents par les pairs sont faites approximativement chaque deux semaines.

- L'auteur poste la contribution à réviser via la *mailing list*. German mentionne que le document doit contenir:
    - Le nom et le courriel de l'auteur;
    - La date;
    - La période de la revue par les pairs,
    - Le statut : ici, les statuts sont : «approuvé», «rejeté», ou «en attente»;

  - La liste des participants de la revue : personnes qui évalueront le document proposé, et qui par la suite l'approuveront ou le rejetteront;
  - La liste des exigences erronées et celles non erronées.
- La revue par les pairs se fait sur une période minimum d'une semaine
- Après la période de la revue, la GEP (*GNOME Enhancement Proposal*) accepte ou rejette la contribution soumise. Un rapport de cette revue est fait et il contient la liste des exigences acceptées et celles à revoir, ainsi que les détails des discussions. S'il y a des exigences mal définies, la revue continue pendant 10 jours, suivie d'une phase de validation du document durant 4 jours. Si la contribution est acceptée par la GEP, une validation de cette contribution est faite sur une période de 4 jours environ, puis les mainteneurs de ce module opèrent par vote pour l'approuver ou non. Si elle est approuvée, les mainteneurs la mettent sur la liste des contributions approuvées. German mentionne également qu'une majorité de 2/3 des votes est requise pour approuver une contribution.

3.2.4 Problèmes de qualité

Vu la grande croissance des logiciels libres [13, 100, 103] et le fait que des produits de logiciels libres sont d'une qualité comparable, voir supérieure à ceux d'un développement fermé [67, 108], les logiciels libres sont considérés comme une alternative viable, attirant non seulement des volontaires, mais également de plus en plus d'entreprises utilisatrices de F/OSS [60]. Cependant, cette qualité élevée dans les logiciels libres n'a pas été une chose facile. Les communautés de logiciels libres en pleine ascension ont été confrontées à certains problèmes tels que :

- Le contrôle de la maintenance à long terme, ainsi que la cohérence du système logiciel à échelle [108]. Le défi est de pouvoir garantir des niveaux acceptables de qualité [108] ;

- Le manque de documentation ainsi que les problèmes de coordination et de communication [92];
- La gestion des configurations ainsi que le code non supporté [92];
- L'attraction des volontaires et le fait que les utilisateurs ne savent pas comment rapporter les bogues [91];
- L'acceptation des logiciels libres dans l'industrie [18, 31, 39, 108]. Ce problème a été un défi considérable dans les F/OSS. L'acceptation des F/OSS dans l'industrie et par les utilisateurs se traduit à la fois :
    - **Sur le plan économique** [18, 39]. Les logiciels libres empiètent sur le territoire des sociétés capitalistes, attirant de plus en plus d'utilisateurs. Ainsi, ils entrainent un nouveau modèle de production pour certaines entreprises, désirant se faire une part de marché : le style *bazar* [18, 102] qui est relativement moins coûteux. Les F/OSS souffrent également d'un problème d'attraction de volontaires [92].
    - **Sur le plan de la sécurité, de confidentialité des données et de fiabilité**. Face à ce problème, certains auteurs, citons, Jai Asundi et Rajiv Jayant [31], Anas Tawileh et Omer Rana [115]; proposent de faire participer les utilisateurs au processus de développement de logiciel. Les logiciels libres se sont montrés d'une sécurité et fiabilité élevées car l'accès au code source permet de rapidement détecter et corriger les erreurs [45]. Cependant, le libre accès au code source constitue également une faille à la sécurité, car il est possible de se retrouver avec des erreurs dans les versions et mises à jour de sécurité, faute à un mauvais système de suivi de versions. Dans ce contexte, des chercheurs ont eu à identifier entre autres problèmes de qualité en logiciel libre, le problème de mise à jour de sécurité [91, 92]. Anas Tawileh et Omer Rana [115], notent l'absence apparente de mécanismes d'assurance qualité dans les FOSS, lesquelles méthodes

sont nécessaires pour garantir la protection des divers intérêts des utilisateurs. Afin de solutionner ce problème, Tawileh et Rana [115], proposent de faire participer les utilisateurs au processus de développement des F/OSS.

3.3 CONCLUSION

En somme, nous remarquons que les communautés de F/OSS sont plutôt structurées en leur sens. La gestion du personnel dans les F/OSS est semblable aux pratiques des entreprises en ce qui concerne les ressources humaines. Cependant, les projets de logiciels libres et Open source sont reconnus pour leur difficulté à définir correctement l'état du travail à faire, leur manque de documentation, leur manque de processus de déclaration des risques de projet, leurs tests informels de système, leur manque de méthodologie d'assurance qualité, ainsi que leurs problèmes en gestion de configurations et de planification de projet. Malgré leur organisation plus ou moins structurée et leur manque des pratiques de qualité normées, nous trouvons cependant des similitudes avec leurs pratiques de qualité et celles normées. Nous citons de ce fait :

- L'initiation d'un projet : l'état du travail à faire pour chaque projet libre dans certaines communautés de F/OSS, est similaire à celui normé, en ce sens qu'il existe un groupe qui se charge de donner une vision du projet et qui définit un calendrier de livrables/révisions que le projet doit respecter. Cependant, ce comité travaille le plus souvent avec les programmeurs dans les projets de F/OSS au lieu des clients comme dans le cas d'un développement fermé, car pour les F/OSS, se sont les développeurs qui définissent les exigences d'un projet. Cependant, les développeurs peuvent travailler en collaboration avec les utilisateurs du produit pour plus de fiabilité.

- Le processus de revues par les pairs : il n'existe aucune autorité supérieure dans le processus de revue des pairs dans les F/OSS, ce qui est semblable aux pratiques industrielles. Il y a un auteur qui définit la contribution à réviser et un délai prévu pour la revue. Chacun des développeurs, participant à la revue dans les F/OSS, respecte ce délai, mais de manière **asynchrone**, car ils sont répartis à travers le monde. Par ailleurs comme différence, le nombre de participants à la revue par les pairs dans les projets de F/OSS dépasse celui prévu par le processus de revue par les pairs opéré en entreprise et prescrit en génie logiciel. À ce niveau, un des aspects avantageux des F/OSS, consiste à une détection et correction rapide des défauts du produit logiciel, et ce, le plus tôt possible.

CHAPITRE IV

MÉTHODOLOGIE

4.1 INTRODUCTION

Les F/OSS connaissent une popularité de plus en plus grandissante depuis la concrétisation du concept de logiciel libre. Ainsi ils sont de plus en plus présents sur la place du marché, rivalisant avec les logiciels propriétaires. Depuis le lancement du concept de logiciel libre en 1983 par Stallman, le style de développement utilisé dans les projets libres suscitait bon nombre d'interrogations du coté des utilisateurs et des entreprises commerciales, à savoir : est-ce que les logiciels libres sont fiables? Les communautés de F/OSS respectent-elles les normes de l'industrie? Et bien d'autres encore. Cependant, comme mentionné dans le chapitre précédant, malgré les problèmes encourus dans les F/OSS et leur processus non rigoureux au sens classique du génie logiciel, il n'en reste pas moins que certains F/OSS sont d'une qualité élevée, généralement les plus populaires, nous citons, Linux, Mozilla, Apache, GNOME. Les logiciels libres sont donc vus comme une solution, un avenir, tant pour de nombreuses PME développant du logiciel et voulant sauvegarder des coûts de licence propriétaires et/ou de services vendus par les entreprises commerciales (déploiement, administration, maintenance, mise à jour de logiciel), que pour des utilisateurs désireux d'apprendre ou de se faire de l'argent. De plus, les logiciels libres sont présentés comme une alternative aux programmeurs face à la rigueur

procédurale des méthodes traditionnelles de développement et ils sont pour ce dire, incontournables de nos jours pour la bonne marche de l'Internet.

4.2 RAPPEL SUJET DE RECHERCHE

Étant donné que le concept de logiciel libre nous semblait un peu plus ambiguë, le travail présenté dans ce mémoire nous permettra d'avoir une idée claire et précise du concept de logiciel libre, et de démystifier la culture, les valeurs, la philosophie et même les principes soutenant le monde du libre. Ce travail de recherche est une étude principalement analytique, qui consiste à recueillir des informations sur le développement libre, ceci dans le but premièrement d'identifier les différences entre le développement de logiciels libres et le développement classique de logiciels, et enfin de voir si la norme ISO/IEC 29110 peut s'appliquer au style de développement de logiciels libres compte tenu de ses valeurs et de sa culture. À cet effet, nous nous concentrons particulièrement sur de grands projets libres à savoir : Linux, Mozilla, GNOME, Apache, car ce sont les plus répandus, ceux connaissant un réel succès et maintenant un niveau élevé de qualité dans les produits développés. Ces projets de F/OSS sont des projets ayant une communauté de développement de taille moyenne à très grande.

4.3 ANALYSE, OUTILS ET ÉLÉMENTS DE L'ÉTUDE

4.3.1 Aspect global de l'étude

Le principal outil pour mener à bien cette étude est la norme ISO/IEC 29110. De plus, afin de recueillir des informations pour mieux appréhender le concept de logiciel libre, de même que tout ce qui l'entoure (valeurs, principes, philosophie, culture) et

les différents processus de développements utilisés, nous avons eu recours aux : wikis, site web des projets F/OSS retenus, travaux passés et actuels menés sur les F/OSS. L'accent de cette étude porte sur le processus de développement de logiciel libre, question d'évaluer si la norme ISO/IEC 29110 pourrait soutenir le développement de logiciels libres et dans quel cas elle pourrait améliorer leur processus de développement.

4.3.2 Recadrage : outils et éléments utilisés

En vue de relever les différences entre le développement libre et le développement classique de logiciel, comme déjà mentionné ci-dessus, nous utilisons la norme ISO/IEC 29110 comme principal outil pour l'évaluation du processus de développement de logiciel libre. En plus de la norme ISO/IEC 29110, nous avons développé des tableaux qui sont des récapitulatifs des informations nécessaires à l'analyse ou l'observation des processus libres, lesquelles informations sont relatives aux pratiques utilisées dans chacune des activités/processus de développement définis par la norme ISO/IEC 29110. Le gabarit de ces tableaux est représenté par la figure 4.1.

Nom de l'activité: ----------

Pratiques définies par ISO/IEC 29110		**Analyse de la réalisation de l'activité et identification des rôles et des artefacts pour chaque activité**											
Référence de la tâche	Description de la tâche	Parfaitement exécutée	Partiellement exécutée	Non exécutée	Aucune connaissance de l'existence de cette tâche	Artefacts ou Commentaires	CUS	AN	DES	PR	TL	PM	WT

Légende:

CUS: Client
AN: Analyste
DES: Designer
PR: Programmeur
TL: Leader technique
PM: gestionnaire de projet
WT: équipe de travail

- Tâches et sous-tâches de chaque activité définie par la norme ISO/IEC 29110
- Critères d'analyse
- Notes indiquant comment est réalisée chaque tâche de l'activité de développement libre et quels sont les rôles associés à chacune des tâches

Figure 4.1 Gabarit du tableau d'analyse

La réalisation de notre méthodologie d'analyse, nécessite d'analyser les éléments suivants:

- **Les activités :**

Une activité est une succession de tâches relatives à la réalisation d'un processus. En génie logiciel, une activité sert d'élément de suivi et de planification. Pour une activité donnée, il y a un ou plusieurs rôles qui lui sont assignés. Ces rôles sont généralement fonction des tâches de l'activité à réaliser. Une activité selon ISO/IEC 29110 [73] consiste en un ensemble de tâches d'un processus cohérent. Une activité est donc un processus du cycle de vie de logiciel qui encadre plusieurs tâches.

- **Les rôles :**

Un rôle représente la fonction et les responsabilités qu'ont les membres d'une équipe. Une personne peut porter plusieurs chapeaux ou rôles, dépendamment de l'activité à exécuter.

- **Les artefacts :**

Les artefacts sont des documents que l'on trouve en entrée et en sortie d'une activité. Les artefacts représentent donc des produits de travail (produits ayant subit des transformations, même minimes) de processus. Sous l'influence du faux-ami anglophone *Artifact*, le mot artefact est parfois employé pour désigner de manière générale un produit ayant subi une transformation, même minime, par l'homme et qui se distingue ainsi d'un autre provoqué par un phénomène naturel [4]. De ce fait, les produits en entrée d'une activité, sont des produits nécessaires à son exécution. Les produits en sortie d'une activité sont les résultats de cette activité (ici, ce sont des produits ayant été générés après l'exécution de l'activité).

Ces éléments ont été retenus pour l'analyse du processus de développement libre car ils sont incontournables pour le bon fonctionnement des différents processus du cycle

de vie de logiciel. Il est question ici de faire une correspondance ou comparaison des pratiques dans le processus de développement de logiciel libre avec celles définies par la norme ISO/IEC 29110, en s'appuyant sur les éléments cités ci-dessus.

4.4 PRÉSENTATION DE L'APPROCHE UTILISÉE

Afin d'atteindre nos objectifs de recherche, comme déjà mentionné lors du précédent paragraphe, la norme ISO/IEC 29110 est utilisée comme principal outil appuyant notre étude.

Rappelons que la norme ISO/IEC 29110 est une norme prescrite pour un profil de TPOs (Très Petite Organisations), en anglais VSEs (*Very Small Entities*), constituées au **maximum de 25 personnes**. Elle présente les processus minimaux de développement que doivent respecter les TPOs. Elle est présentée en détails au chapitre 6.

Le premier constat que nous faisons de cette norme est qu'elle est réservée pour de très petites entreprises, organisations, projet et mêmes de petites équipes d'au plus 25 personnes au sein de grandes entreprises. Or les communautés de logiciels libres abritent généralement des centaines, voire des milliers d'individus. Il convient donc de se poser la question suivante : Est-ce que la norme ISO/IEC 29110 pourrait soutenir le développement de logiciels libres et pourrait aider les communautés du libre ?

En nous attardant seulement sur le cadre conceptuel de la norme, ici pour quel type d'entreprise ou projet elle a été prescrite, nous en viendrons à la conclusion immédiate que cette norme ne peut s'appliquer aux logiciels libres, et donc, elle ne pourrait soutenir leur développement compte tenu du fait que les F/OSS sont

constitués de centaines, voire de milliers de personnes réparties à travers le monde. Cette idée est plutôt faussée de par la structuration ou organisation des communautés de F/OSS. De notre observation des communautés de F/OSS, nous constatons qu'elles sont divisées en de petites entités ou sous-projets. Chaque sous-projet au sein du projet global de F/OSS (c'est lui qui représente la communauté de F/OSS), possède ses propres membres ou participants, et représente en lui-même une petite communauté libre, et il est lui aussi divisé en de plus petites entités de développement. Cette observation des F/OSS, semble globalement soutenir l'idée que la norme ISO/IEC 29110 peut s'appliquer aux F/OSS : concept de petites équipes de développement au sein d'une grande organisation, mais elle n'est pas encore effective.

Outre la précédente observation de la structuration des F/OSS et du fait que la norme ISO/IEC 29110 est prescrite pour un profil de TPOs, lors de la lecture approfondie de la norme ISO/IEC 29110, nous avons relevé une note très importante. Cette note stipule que la norme ISO/IEC 29110 n'est pas destinée à empêcher ni à décourager l'utilisation des pratiques des processus du cycle de vie qu'elle décrit, pour des organisations ou entreprises plus grandes (voir page 1 de la norme ISO/IEC 29110).

« *The life cycle processes described in the set of ISP*[15] *and Technical Reports are not intended to preclude or discourage their usage by organizations bigger than VSEs.* » [73].

Et donc, cette phrase n'exclut pas l'idée selon laquelle la norme ISO/IEC 29110 pourrait s'appliquer à des organisations plus grandes que les VSEs, dans notre cas les communautés de logiciels libres. Hors les VSE sont des entreprises, des organismes, des départements au sein de grandes organisations, ou des projets comportant au plus 25

[15] ISP: Software Engineering International Standardized Profile [73].

personnes. Compte tenu de la structuration des F/OSS vu plus haut, nous retrouvons donc l'aspect de petites entités ou départments au sein d'une très grande organisation. Ainsi, dans un cadre plus pratique d'une application effective de la norme aux F/OSS, nous recommandons que lors de la conception du développement libre, que leur développement soit organisé en pools dont le nombre est inférieur ou égale à 25 personnes. Pour que cela puisse être réalisable dans les F/OSS, tout en sachant déjà qu'un projet de F/OSS est divisé en de plus petits projets eux-mêmes divisés en de plus petites entités, les gestionnaires de projets dans les F/OSS pour chaque module du projet qu'ils maintiennent, doivent s'assurer que chacune des parties de développement de leur module de projet ne contient pas plus de 25 personnes. De cette manière la théorie de la norme selon laquelle elle est applicable pour des TPOs (organisation de développement d'au maximum 25 personnes) sera parfaitement respectée. Cela nous a donc motivé à poursuivre notre étude sur la qualité dans les F/OSS et de voir effectivement si la norme ISO/IEC 29110 peut s'appliquer aux F/OSS. Si oui, comment cela se fait? Par la suite, évaluer si la norme ISO/IEC 29110 pourrait améliorer, même en partie, le développement libre. Auquel cas, quelles seront les éventuelles améliorations du processus de développement des F/OSS que nous aura procuré notre application de la norme aux F/OSS. À cet effet nous nous sommes fixés certains objectifs de recherche (voir chapitre 2 – section 2.3).

De plus, l'accent de cette étude porte sur la qualité des produits logiciels libres développés, précisément sur le processus de développement de logiciels libres. Par ailleurs, la norme ISO/IEC 29110 présente les processus minimaux de développement en génie logiciel qui doivent se retrouver dans un cycle de développement quelconque de logiciel. Bien que la gestion de projet ait un impact sur la qualité du logiciel, nous avons choisi de nous limiter au processus d'implémentation de logiciel (*Software Implemention Process*) décrit dans la norme à partir de la section 4.3 (voir figure 6.3 au chapitre 6) car les projets de logiciels libres n'ont généralement pas un processus de gestion de projet [94] tel que celui connu en génie logiciel (pour les

détails, voir section 4.4.1) et que les projets libres ont tendance à aller directement à la programmation [26]. De ce fait, la lecture et la compréhension de la norme sont des conditions préliminaires de notre étude. Chaque activité de la norme sera étudiée en détails, en s'attardant le plus sur les tâches car elles serviront pour faire l'analyse des pratiques utilisées dans développement de logiciels libres avec celles de la norme ISO/IEC 29110, lesquelles représentent ici le standard que nous utilisons pour le cycle de vie de logiciel en ingénierie. Cette première analyse nous permettra de voir en quoi le développement de logiciel libre se distingue de celui en génie logiciel, ceci en plus des connaissances déjà acquises au travers de certains travaux de recherche menés – différences entre le développement de F/OSS et le développement classique [26, 94]. Par la suite, nous évaluerons la norme ISO/IEC 29110 par rapport aux F/OSS, et éventuellement à partir des résultats obtenus, verrons comment la norme ISO/IEC 29110 pourrait aider à améliorer la performance du processus de développement libre. Une description du processus d'implémentation de logiciel tel que prescrit par la norme ISO/IEC 29110 est présentée au chapitre 6 – section 6.1.2.2 lors de la présentation de la norme. Voir particulièrement la figure 6.3 : Diagramme d'implémentation de processus, dont l'aperçu est présenté ci-dessous.

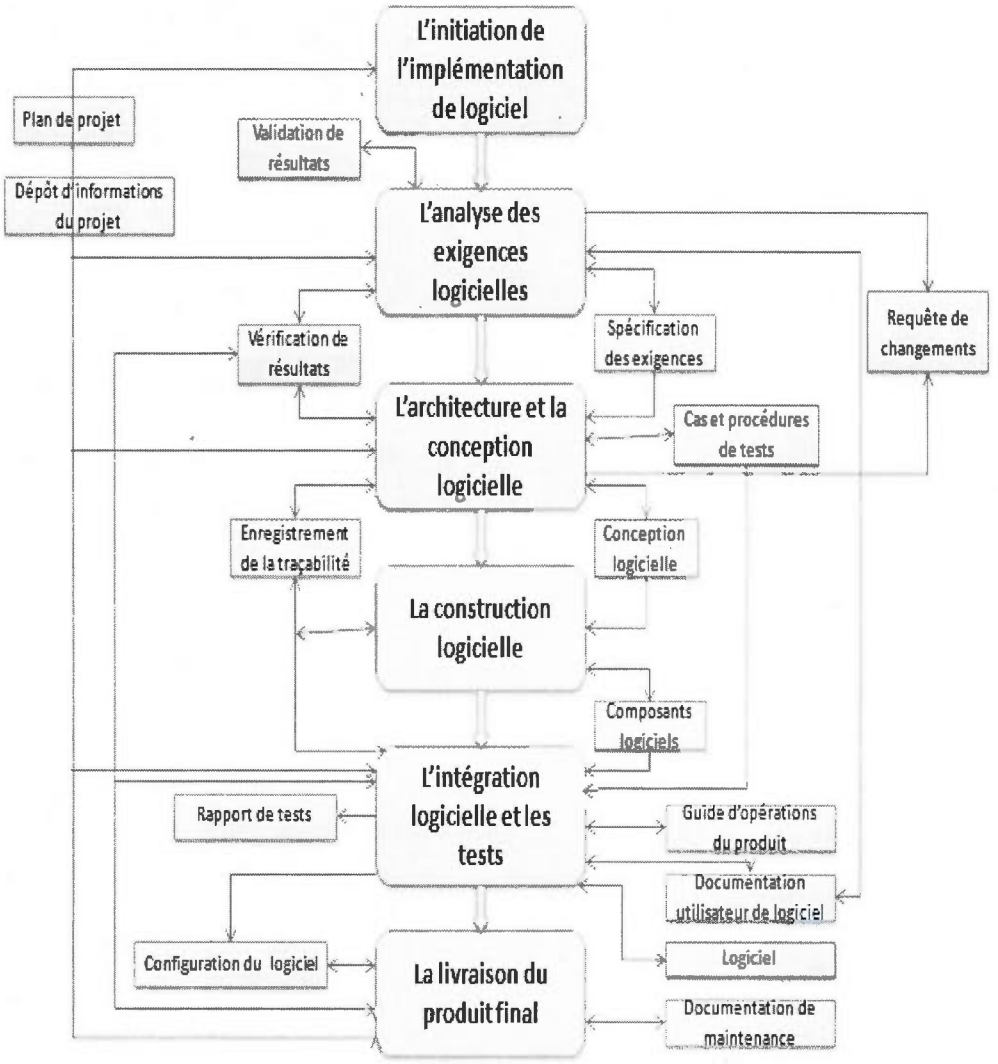

Figure 6.3 Diagramme d'implémentation de processus traduit de [73]

De façon plus précise, afin d'analyser le processus de développement de logiciels libres, nous avons procédé à différentes comparaisons. Nos approches de comparaison sont divisées en deux principaux exercices :

1- L'évaluation du processus de développement de logiciel libre, en vue de déterminer les différences entre les processus libres et ceux en génie logiciel;

2- L'analyse de compatibilité des activités de la norme ISO/IEC 29110 avec les F/OSS, en vue d'identifier les activités de la norme compatibles aux F/OSS, et donc, qui pourraient aider à améliorer le processus de développement libre.

4.4.1 Méthodologie d'analyse du processus de développement libre

Notre méthodologie d'analyse du processus de développement de logiciels libres peut ainsi être représentée de façon générale par la figure suivante :

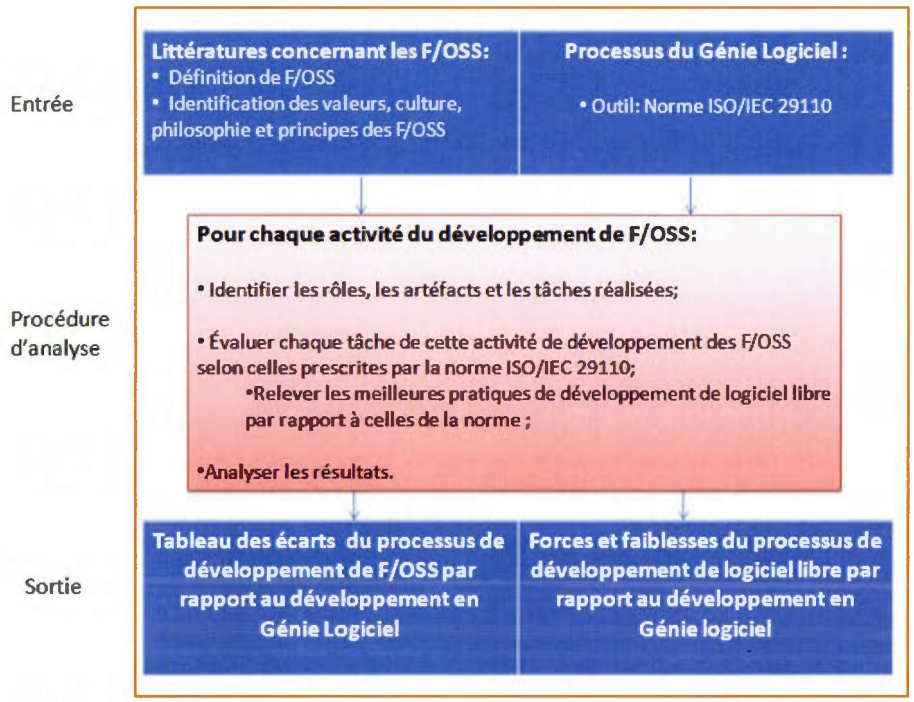

Figure 4.2 Analyse générale du processus de développement de logiciel libre

L'approche que nous utilisons pour relever les différences entre les processus de développement libre et le développement classique de logiciel est une étude

analytique du développement libre en se basant principalement sur la norme ISO/IEC 29110, particulièrement sur le processus d'implémentation de logiciel.

Nous n'avons pas jugé nécessaire de faire une étude du processus de gestion de projet qui est décrit dans la norme ISO/IEC 29110 au niveau de la section 4.2, car les projets de logiciels libres n'ont généralement pas un processus de gestion de projet [94] tel que celui connu en génie logiciel et que généralement, ils vont directement à la programmation [26]. La gestion de projet en logiciel libre, tel que définie par Mockus, Fielding et Herbsleb [94] est: « l'initiation et la planification d'un projet, de manière à attirer le maximum de personnes ».

Il semble plus qu'évident que l'analyse du processus de développement de logiciels libres doit se faire par une observation préalable des pratiques utilisées dans les projets de logiciels libres. Dans ce contexte, nous avons limité notre étude aux projets suivants : Linux, Mozilla, Apache, GNOME; car ce sont de très grands et populaires projets de F/OSS. La première phase de notre étude consistera à présenter le processus de développement actuel de logiciels libres (voir chapitre 5) compte tenu de la revue de littérature faite dans le chapitre 3, laquelle se base sur : les anciens et récents travaux menés sur les F/OSS, les informations trouvées sur les wikis et les sites web des projets logiciels libres sélectionnés et précédemment cités. La deuxième phase de notre étude consiste en une évaluation ou comparaison de différentes activités du développement libre avec les activités décrites dans la norme ISO/IEC 29110 (voir chapitre 6). À cet effet, pour chacune des activités basiques du cycle de vie de logiciel (précisément : *activité d'initiation et d'implémentation de logiciel, activité d'analyse des exigences, activité d'architecture et de conception, activité de construction du logiciel, activité d'intégration et tests, activité de livraison du produit*), leurs tâches et leurs sous-tâches seront observées en relation aux tâches suivantes (voir figure 4.2) :

- Élaboration des critères d'analyse;

- Identification des rôles requis et des artefacts générés lors de la réalisation de ce processus libre et les comparer aux pratiques de développement de logiciel définies par la norme. Cela permettra de relever les écarts entre le développement libre et celui en génie logiciel;
- Analyse des résultats.

Suite à la lecture de la norme ISO/IEC 29110, nous avons ainsi tiré les éléments ci-dessous, lesquels sont importants pour observer les écarts et les correspondances de processus dans le monde du logiciel libre conformément à la norme. Ces éléments ne sont ni plus ni moins que les éléments d'analyse, mentionnés dans la section 4.3.2, cependant, avec plus de précision. Ces éléments sont les suivants :

- L'existence d'une activité;
- Les tâches et sous-tâches d'une activité;
- Les rôles requis pour l'exécution de chaque tâche et sous-tâche;
- Les artefacts produits à la sortie de chaque tâche. Ces artefacts démontrent que ces tâches ont été faites.

4.4.2 Compatibilité de la norme avec les logiciels libres

Nous avons choisi une méthode basée sur des **critères de compatibilité** aux F/OSS, car selon nous, elle semble être une méthode fiable et efficace pour évaluer la compatibilité de la norme ISO/IEC 29110 selon le contexte des F/OSS. Compte tenu que la norme ISO/IEC 29110 comporte en tout cinq activités pour le processus d'implémentation de logiciel, selon le nombre d'activités de la norme qui seront compatibles aux F/OSS, nous pourrons donc aisément déterminer le pourcentage de compatibilité de la norme ISO/IEC 29110 à la culture, aux valeurs et même à la philosophie des F/OSS.

Pour la détermination des activités de la norme qui sont compatibles et celles non compatibles avec la philosophie des F/OSS et les considérations autour des F/OSS, nous tiendrons non seulement compte du mode de développement utilisé en logiciel libre, mais également des valeurs soutenant le comportement des participants aux F/OSS. Nous ferons ainsi des tableaux comparatifs de chacune des activités du processus d'implémentation de la norme ISO/IEC 29110 avec celles du logiciel libre, et ce, tout en tenant compte des valeurs du logiciel libre. Pour parvenir à ces différentes comparaisons de processus[16], nous avons premièrement identifié nos critères de compatibilité.

Afin d'identifier les différents critères de compatibilité, nous avons étudié en détails quelques projets de F/OSS dans les communautés Linux, Apache, Mozilla et GNOME. Le but étant de faire ressortir les similitudes ou synthèses des pratiques communes à chacun des sous-projets retenus. Ces similitudes constitueront les différents critères de compatibilité. Les sous-projets retenus à cet effet sont : le noyau Linux, Apache http server, GNOME (car il est lui-même un sous-projet du projet GNU/Linux), GCC, Firefox. Ils ont retenu notre attention car ils sont très populaires dans leurs communautés respectives, ainsi qu'en logiciel libre en général. Ainsi, ils regorgent d'une base d'informations assez dense.

Cet aspect de compatibilité aux F/OSS est très intéressant, du fait qu'il nous permettra d'introduire une façon dont la norme ISO/IEC 29110 pourrait être appliquée aux logiciels libres afin d'améliorer leur processus de développement, compte tenu des différentes valeurs encourues dans les F/OSS.

---

[16] Dans le contexte libre, une activité de développement est perçue comme un processus. Par contre dans le cadre de la norme ISO/IEC 29110, il y a deux principaux processus : le processus de gestion de projet et le processus d' implémentation de logiciel. Chacun de ces processus est constitué de multiples activités. Dans le cadre de ce travaille de recherche, nous employons souvent le terme «**processus**» pour désigner une «**activité**» de développement.

La méthodologie d'analyse que nous utilisons à ce niveau est résumée par la figure 4.3.

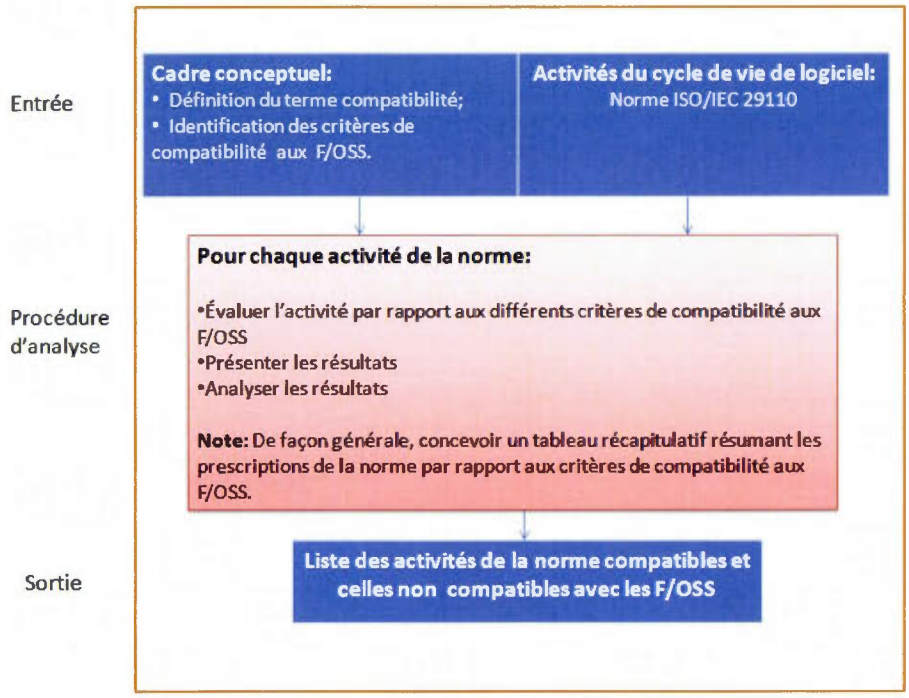

Figure 4.3 Analyse de compatibilité de la norme avec les logiciels libres

4.5 CONCLUSION

La méthode présentée dans ce chapitre nous permettra d'atteindre nos objectifs de recherche. Nous pourrons ainsi, par le déroulement et l'exécution de cette démarche de recherche, identifier premièrement en quoi le développement de logiciel libre se distingue du développement en génie logiciel. Deuxièmement, déterminer les

activités de la norme ISO/IEC 29110 qui sont compatibles et celles non compatibles avec les logiciels libres.

CHAPITRE V

ANALYSE DU PROCESSUS DE DÉVELOPPEMENT LIBRE SELON L'OBSERVATION DES F/OSS

5.1 INTRODUCTION

Le présent chapitre consiste à faire une analyse ou synthèse des pratiques utilisées dans le processus de développement libre compte tenu de la revue de littérature faite au chapitre 3. Cette analyse se concentre sur les projets Linux, Mozilla, Apache et GNOME.

La compréhension du processus actuel de développement de logiciels libres, nécessite de garder à l'esprit le cadre conceptuel de cette analyse, consistant à l'identification du type de développement utilisé dans les F/OSS (chapitre 3 - section 3.1.4) et à l'identification des outils et des éléments requis à notre analyse (chapitre 4 - section 4.3.2). Dans ce chapitre, le processus actuel de développement libre sera présenté.

5.2 ANALYSE DU PROCESSUS DE DÉVELOPPEMENT LIBRE

5.2.1 Présentation du processus de développement actuel

La présente section consiste à faire une présentation des pratiques actuelles utilisées dans le cycle de vie de logiciels libres en se basant sur la norme ISO/IEC 29110 (voir figure 6.3).

5.2.1.1 Initiation d'implémentation du logiciel

De notre observation des F/OSS, les projets de F/OSS ne proviennent généralement pas d'un besoin explicite des clients, mais plutôt du besoin d'un programmeur. Néanmoins, le client peut également être un utilisateur[17]. Ce contexte conduit dans le monde du libre à une connotation de l'activité d'initiation d'implémentation du logiciel autre que celle connue en génie logiciel. En logiciel libre, l'initiation d'implémentation d'un projet est vue au travers de la proposition de projet faite par un développeur et qui a été postée via la *mailing list* ou sur le site web de sa communauté de logiciels libres.

De par le mode de développement utilisé dans les communautés de F/OSS, il n'existe pas une manière formelle de capturer, d'analyser, de valider les spécifications d'un projet. De plus, selon Mockus, Fielding et Herbsleb [94], entre autres différences du développement libre et du développement traditionnel de logiciel, il n'existe pas de plan de projet en logiciel libre, ni de calendrier de révisions et des listes de livrables.

---

[17] Ce sont des personnes utilisant un logiciel de type libre et contribuant aux projets de logiciels libres par des rapports de bogues, des suggestions de fonctionnalités [19, 102]. Dans les communautés de logiciels libres, les utilisateurs sont dans la plupart des cas des développeurs.

Particulièrement dans Apache, pour pallier à cela, les revues sont faites couramment et de manière répétitive [105]. A l'instar du projet Apache, Linux n'utilise généralement pas de plan de projet dans lequel sont définis les calendriers de révisions. Les revues sont également récurrentes et se font de manière répétitive, telle la règle de Linus Dorval - « Distribuez tôt, mettez à jour souvent » [102]. Ce résultat est toutefois contredit dans GNOME [60] et Mozilla [104] en ce qui concerne typiquement les calendriers de révisions et les listes des livrables. Les projets Linux, Apache, Mozilla, GNOME, utilisent la modularité du code comme pratique de qualité [26, 36, 60, 104, 105]. Cette modularité du code de base permet de simplifier le développement et de facilement détecter et de corriger les bogues. Chacun des différents modules du projet, possède ses propres objectifs et sa propre communauté (en occurrence son site web, sa liste de diffusion, ainsi que ses propres membres). Dans chaque module, les propositions ou descriptions de projets sont écrites par un membre de l'équipe de travail de base et sont soumises aux autres membres pour révision (incorporation des commentaires) [60, 94] et/ou pour voir si ce projet vaut la peine d'être réalisé [19]. L'équipe de développement de base en logiciel libre est composée de développeurs de confiance qui s'assurent que le projet marche et veillent à ce que le projet attire de nouvelles personnes. Ainsi, une proposition de projet est faite par un développeur et mise à la disposition du reste des membres de sa communauté au travers de la *mailing list* ou sur le site web de sa communauté, car ce sont les développeurs de F/OSS qui implémentent leurs systèmes et définissent certaines fonctionnalités nécessaires à ce système. Le développeur peut cependant travailler avec les utilisateurs, comme le proposent Anas Tawileh et Omer Rana [115], afin de faciliter l'acceptation de ce projet en entreprise.

Les gestionnaires ou mainteneurs de projets sont les chefs ou les leaders techniques dans les communautés de F/OSS, dont les rôles selon Lerner et Triole [3] sont de :

- Fournir la vision du projet;

- Diviser le projet en plusieurs parties dans lesquelles les individus peuvent s'attaquer aux tâches indépendantes;
- Attirer les développeurs au projet;
- Garder le projet en un bloc et prévenir la bifurcation.

De même, en 2007, Aberdour [26] a également eu à identifier le rôle de l'équipe de base ou chef de projet :

« The core team must be small. As well as performing the bulk of the coding activity, the core team will exert control over the core system to maintain high modularity. They integrate only high-quality code, determine and follow the project roadmap, and maintain a fast, iterative release cycle », tiré de [26].

De plus, de notre observation, les chefs de projet travaillent avec l'équipe de travail (ici les développeurs), afin que le projet respecte les dates prévues pour la révision d'une version précédente du projet, ainsi que les exigences des utilisateurs [60, 94]. Cette façon de procéder est quelque peu différente de GNOME, car dans ce projet, il existe une équipe de révision ou d'assurance qualité dont le rôle est principalement de planifier et de coordonner l'ensemble du projet, de manière à ce que le projet suive le calendrier des révisions (revue de pairs) proposées [60]. De plus, cette équipe de révision est chargée d'élaborer, en coordination avec les chefs de projet, les calendriers de livraison/révision pour chacun des différents modules et la planification de tout le projet. Une fois que les chefs ou gestionnaires de projets ont validé tous les changements dans le calendrier de révisions et dans la liste des livrables, ils donnent les différentes orientations du projet en le divisant en modules et postent les tâches à réaliser pour chacun des modules sur le site web de la communauté. Par la suite, les développeurs n'auront qu'à choisir les tâches du module sur lesquelles ils désirent travailler en relation avec leur expertise. Les mainteneurs de chaque module supervisent le développement de leur module,

coordonnent et intègrent les contributions des autres développeurs de leur module [60, 104]. Nous notons cependant que plusieurs des chefs de projets en F/OSS sont des employés des compagnies qui financent le projet de F/OSS [30, 60, 87].

5.2.1.2 Analyse des exigences

En entreprise, une bonne compréhension des exigences du logiciel, permet de construire le produit logiciel avec le moins d'anomalies liées aux besoins du client. Par contre, de l'observation faite sur le développement de F/OSS, nous remarquons que les communautés des F/OSS ne semblent visiblement pas adopter, ni pratiquer le processus classique d'exigences logicielles [106]. Cependant, les logiciels développés dans certaines communautés libres se sont montrés viables, extrêmement fiables, d'une qualité élevée et grandement utilisés dans leurs communautés d'utilisateurs respectives. Dans cette section, il est question de déterminer quelles pratiques sont utilisées dans le processus d'exigences pour les F/OSS.

La proposition de travail est faite par un développeur. Ce dernier définit les exigences du logiciel et les poste habituellement via la *mailing list* ou sur le site web de sa communauté de F/OSS, ou dans les *documents de spécification des exigences*[18], disponibles pour des revues, élaboration ou raffinement ouverts [107]. La communication des exigences diffère tout de même d'une communauté à une autre [106]. Étant donné que les participants aux projets de F/OSS communiquent à travers l'Internet, les documents postés via l'infrastructure du libre doivent être de type neutre (de préférence en PDF) pour qu'ils puissent être lisibles par tous les participants au projet. Les exigences sont généralement des affirmations suite à des

[18] Document des exigences du logiciel, dans le cas où les exigences ne sont pas sous forme de messages filetés (*mailing list*) ou des discussions sur le site web de la communauté.

discussions privées ou publiques faites par courriel, des artefacts de logiciels ad hoc (tels que des *fragments de code source*[19] inclus dans le message) et des mises à jours du contenu du site web de la communauté [107], des artefacts logiciel post-hoc (fonctionnalité du projet final à rajouter au module) [60]. L'analyse et spécification des exigences de F/OSS sont donc des activités implicites [107]. Comme le mentionne Walt Scacchi [107], les exigences de F/OSS apparaissent de manière routinière :

« *They routinely emerge as a by-product of community discourse about what its software should or shouldn't do and who'll take responsibility for contributing new or modified system functionality [...] More conventionally, requirements analysis, specification, and validation aren't performed as a necessary task that produces a mandated requirements deliverable* » [107].

Dans les communautés de F/OSS, les leaders techniques (dont les différentes appellations sont : *Committer* dans Eclipse [47] ou mainteneurs de projets[20] dans GNOME [60], développeurs de base dans Apache, connus sous l'appellation de Apache Group(AG) [94] ou *committers* [9]), l'équipe de travail et quelques fois des utilisateurs, participent activement à la réalisation des exigences du logiciel afin qu'elles deviennent claires et précises. Nous notons cependant que le fonctionnement d'une communauté à une autre, diffère en fonction de son organisation, de son infrastructure et des règles qui régissent son fonctionnement. Quelques fois, les leaders techniques peuvent avoir recours à des informations de projets antérieurs pour voir si le nouveau projet possède des similarités avec ceux ayant déjà été réalisés [60]. Cette façon de procéder est commune aux grands projets de F/OSS,

---

[19] Plus connus sous l'appellation de « *Patch* ».

[20] Ils peuvent aussi jouer le rôle de gestionnaires de projets. Il en existe pour chaque module du projet et pour le projet global.

particulièrement Linux, Mozilla, GNOME, Apache et même Eclipse. Nous notons que dans le projet GNOME, seuls les leaders techniques qui sont eux-mêmes des développeurs, définissent les exigences du logiciel. Cependant, leurs décisions peuvent être influencées par les programmeurs [60].

De plus, les leaders techniques travaillent avec les développeurs pour la planification du projet afin de **décider de la faisabilité du projet** à partir des spécifications et également compte tenu de l'attraction que ces projets auront sur le public [60, 94]. Une fois cela fait, si les exigences ont été approuvées par la totalité du groupe de leaders techniques, les mainteneurs de projet ou leaders techniques les mettent sur la liste de contributions du projet [60], puis les différentes orientations du projet ainsi que la liste des différentes tâches à réaliser sont postées sur le site web de la communauté ou sur le forum de discussion, par l'un des leaders techniques. Afin d'aboutir à l'approbation des exigences, de nombreuses vérifications (généralement c'est la revue par les pairs qui est faite, et quelques fois des walkthrough [26, 31, 36, 38, 52, 60, 55, 92, 105, 115]) et mises à jour sont faites par l'équipe de travail et leaders techniques [60, 94]. Les exigences dans les F/OSS sont validées par rapport à l'implémentation du logiciel [106]. Les utilisateurs ou les clients ne participent généralement pas à cette phase de vérification et de validation des exigences du logiciel dans les communautés de F/OSS. À cet effet, durant quelques années, les produits logiciels libres ont rencontré des difficultés à être acceptés par les entreprises commerciales. Cela à pousser en 2005, Tawileh et Rana [115] à investiguer le problème et à recommander ainsi la participation active des utilisateurs externes à la communauté, précisément, les entreprises commerciales; pour que le logiciel respecte le plus possible leurs spécifications et exigences. La recommandation de Tawileh et Omer permettrait donc de faciliter l'acceptation du logiciel libre en entreprise. À cet effet, nous avons remarqué que certains grands succès de logiciels libres, par exemple, Apache, Mozilla et GNOME, travaillent avec certaines entreprises commerciales. Par exemple, Netscape emploie des programmeurs pour qu'ils

travaillent à plein temps sur le code de Mozilla, tout comme le font les compagnies IBM, Sun Microsystems et Red Hat [87]. Tel est également le cas dans les projets GNOME [60], Apache [30]. Ces entreprises commerciales contribuent de façon indirecte à développer et à améliorer les produits de F/OSS dont ils ont besoin. Les F/OSS retenus pour toute notre étude ont donc opté de faire participer les utilisateurs aux projets développés. Selon Raymond [102], l'un des principes fondamentaux du développement libre, consiste à : Traiter vos utilisateurs en tant que co-développeurs est le chemin le moins semé d'embûches vers une amélioration rapide du code et un débogage efficace. Cette façon de procéder n'est pas inconnue, ni récente aux F/OSS car elle existait déjà bien avant 1999, lors du développement du noyau Linux [102], et donc, constitue un principe essentiel contribuant à la qualité des logiciels libres. Les utilisateurs de logiciels libres contribuent aux projets en fournissant des *feedbacks* aux programmeurs sous forme de rapport de bogues et de suggestion de fonctionnalités [19, 102].

L'étape de validation des spécifications prendra fin lorsque les leaders techniques posteront les orientations et les tâches du projet à réaliser via la *mailing list* ou sur le site web de la communauté de ce projet et mettront à jour leur infrastructure (le cas échéant. Voir section 3.2.3.2.3 du chapitre 3 pour les détails). S'il y a des changements à effectuer, un document de changements est généré par les leaders techniques et posté sur le site web afin d'en informer tous les participants du projet.

Par la suite, chaque participant au projet, particulièrement chaque développeur choisit la tâche qu'il désire réaliser selon ses préférences et son expertise. Les exigences du logiciel dans les projets de F/OSS ne prennent généralement pas forme suite à une analyse de besoins volontaires, mais plutôt d'un débat via le forum de discussion, dont l'intention n'est pas de faire une analyse de besoins [60].

Une fois les exigences validées et les changements stabilisés, il peut arriver à l'occasion que les programmeurs procèdent à la documentation utilisatrice du logiciel et la publie sur le site web de la communauté pour révision par les leaders techniques avant de sauvegarder la version révisée dans le CVS. Une fois cela fait, le même processus récurrent de mise à jour du site web de la communauté est fait. Néanmoins, s'il existe une documentation du logiciel, elle sera sous forme de page d'aide en ligne. De plus, elle sera stockée dans le référentiel des documents « *baselined* », tâche effectuée par les leaders techniques utilisant les outils suivants : CVS, bugzilla, GNATS, LXR, CCB (voir section 3.3 du chapitre 1), afin de coordonner, superviser et de maintenir le système. Si des changements sont nécessaires, il y a un processus pour les changements qui est suivi. Le CVS sert à la fois de mécanisme centralisé pour la coordination du développement de F/OSS et aussi de contrôle de médiation sur les améliorations, les extensions ou les mises à jour du logiciel, qui seront contrôlées dans les archives [107]. Le contrôle des versions du logiciel (processus de gestion des configurations), est récurrent et commun à tous F/OSS de grande envergure.

Pour finir, compte tenu que toute la documentation dans les F/OSS est en ligne afin de supporter les développeurs et les utilisateurs finaux [106], tous les documents de spécifications des exigences du logiciel (le cas échéant) et de la documentation utilisateur (habituellement sous forme de pages d'aide en ligne) sont intégrés dans le dépôt de documents du projet. Cette étape est mise à exécution par les chefs techniques.

5.2.1.3 Architecture et conception

De notre observation du développement de F/OSS dans Apache, Linux, GNOME et Mozilla; il n'existe habituellement pas de phase explicite de conception globale ou détaillée du logiciel [35, 79, 94, 104]. Le logiciel est construit au fur et à mesure à

travers ces différents modules [104]. Ce processus routinier et itératif, fait effet de phase de conception de logiciel, au cours duquel, l'architecture du logiciel prend forme. Cela est la plupart du temps valable pour des projets logiciels libres partant de zéro, à l'opposé de certains projets F/OSS construits à partir d'une version antérieure de produit. Par exemple, Mozilla a été construit en réutilisant le code source du navigateur Netscape 5 [88, 104] et Apache est basé sur le serveur NSCA HTTPD [88]. Ces deux derniers ont pour base, l'architecture logicielle et les documentations du produit logiciel dont ils héritent. Dans le cas de Mozilla, l'architecture logicielle a été réécrite en partie afin de supporter les nouvelles technologies [104]. Les projets Linux, Apache, Mozilla et GNOME étant de grands projets de F/OSS, ils utilisent donc la modularité du code lors de la phase de conception de logiciel [98]. Quant à la majorité des projets de type F/OSS construits de zéro, en d'autres termes, des F/OSS qui n'héritent d'aucun autre projet, ils sont le plus souvent hébergés sur SourceForge [112].

Jacques Longchamp [88] soutient le fait qu'il n'existe pas de conception logicielle lors du processus de développement de logiciels libres. Cependant, il mentionne une certaine nuance, relatant que la conception du logiciel prend place au tout début du projet[21] pour des projets construits à partir de zéro par un individu ou un petit groupe fermé.

Nous n'avons pas trouvé de documentation relatant comment se fait la conception de logiciel dans les projets de logiciels libres, exception faite de Mozilla. De ce fait, nous ne pouvons donner les détails de ce processus de manière générale sur l'ensemble de ces projets. Nous présenterons donc ci-dessous comment se fait la conception du

---

[21] La conception logicielle prend place au début du projet : particulièrement lorsque la version antérieure du produit, laquelle est quelques fois construite partant de zéro et très souvent en réutilisant ou en améliorant un produit déjà existant, est produite par un individu ou un petit groupe fermé [88].

logiciel dans Mozilla, ainsi que les différents rôles associés et artefacts générés, présentés en détails par Reis et Forbes, dans l'article intitulé : « *An overview of the software engineering process and tools in the Mozilla project* » [104].

Selon Reis et Forbes [104], la conception et les spécifications de l'architecture (architecture originale et la nouvelle architecture de mise en page) de Mozilla sont documentées. Les traitements automatisés et les analyses sont effectués afin d'améliorer la conception et la documentation de code (rapporté par Reis et Forbes des articles de : Brian B. Bramlett – *Code Documentation and Cross-reference* – et Datrix – *Source Code Analysis of Netscape's Communicator 5.0 developement release*). Les problèmes de conception sont abordés dans Mozilla lorsqu'ils surviennent. Tel est également le cas dans la grande partie des processus ad-hoc des exigences : les bogues[22] sont déposés afin de modifier la conception ou l'API d'un composant, puis des discussions ordinaires se font par des commentaires de bogues. La majorité du code des modules de Mozilla possèdent un ou plusieurs composants qui leurs sont associés et présents dans l'outil de suivi de bogues Bugzilla.

« This design is by nature modular, and parts such as the Javascript engine, the runtime libraries, and the framework itself can be reused independently of the browser to develop other products » [104].

À chaque composant de module dans Bugzilla, est lié : un développeur qui est l'auteur du composant et qui joue aussi le rôle de propriétaire de module, et un membre de l'équipe d'assurance qualité, ici un super examinateur, aidant à traiter un bogue. Le propriétaire du module est l'autorité en place qui sélectionne et implémente

[22] Un bogue dans Mozilla se réfère à toute demande de modification déposée dans le logiciel, que ce soit un défaut réel, un accessoire, ou un changement de fonctionnalité. Toutes les demandes de changement et leur mise en œuvre associée ont un numéro unique qui les identifie [104].

des changements dans l'architecture. De plus, ce dernier a le droit d'accepter ou de refuser un changement ayant été soumis. Chaque développeur intéressé par un bogue, s'assigne la tâche et propose une correction. Chaque bogue porte un numéro pour son identification de même que certaines caractéristiques (nom de l'auteur/propriétaire, fichiers qui lui sont attachés, gravité et priorité du bogue, statut, nombre de commentaires), permettant de facilement faire le suivi. Le numéro d'identification du bogue peut être échangé comme le code entre les développeurs[23]. Le propriétaire du module joue également le rôle de propriétaire de composant Bugzilla [15], car, il est l'autorité en place pour ce module. Il définit les cas de tests et s'assure de la bonne intégration des différents composants de son module, conformément à la conception logicielle.

L'organisation de Mozilla [14], la fondation Mozilla, est un groupe chargé du leadership du projet Mozilla et est constitué de 14 personnes[24] sélectionnées dans la communauté, jouant les rôles de gestionnaires ou leaders techniques pour les différents projets de Mozilla. Les membres de l'organisation de Mozilla ont pour tâches de: maintenir le site web, la documentation, la conception de l'architecture et la *release Engineering*. Les différents rôles présents dans la fondation Mozilla sont : propriétaires de module (*Module owner*), super examinateur (*Super reviewer*), propriétaire du composant Bugzilla (*Bugzilla component owner*), pilote d'examen ou révision (*Driver*) et dictateurs bienveillants [15, 104].

Pour l'activité d'architecture et de conception dans Mozilla, le propriétaire du module et le super examinateur travaillent ensemble afin d'effectuer des changements dans la conception du logiciel. Ils permettent à la conception logicielle d'évoluer dans une

---

[23] Pour plus de détails sur la résolution des bogues dans Mozilla, lire l'article de Reis et Fortes [104].

[24] Ce sont des développeurs et membres de la communauté ayant des qualifications particulières, ou qui sont engagés dans le projet [104].

direction particulière. Le propriétaire du module ainsi que le super examinateur ont une bonne connaissance de la conception courante. De plus, chaque développeur dans Mozilla, utilise le CVS, afin d'effectuer couramment des changements ou de contrôler indépendamment le code source, qui lui, est centralisé. En fait, il n'existe qu'une seule image de code source. Ainsi, chaque développeur peut voir les changements opérés par d'autres, et ainsi peut effectuer des tests de régressions sur de grandes proportions des codes sources de modules du projet qui sont intégrés par le biais du CVS.

Des mises à jour sont faites continuellement sur la conception logicielle dans Mozilla, contribuant ainsi au *refactoring* du code source de façon récurrente. Le *refactoring* du code est un processus récurrent, permettant aux développeurs de simplifier les APIs, de supprimer le code non utilisé et d'améliorer la modularité et la lisibilité du code. Ce processus est très souvent utilisé en logiciels libres, car il permet de sans cesse éviter de détruire l'architecture logicielle. En d'autres termes, le *refactoring* du code permet d'éviter de rendre l'architecture logicielle non fonctionnelle, car aucune fonctionnalité supplémentaire n'est rajoutée au logiciel. Le code source du logiciel est tout simplement retravaillé afin d'améliorer sa lisibilité, de simplifier sa maintenance ou de changer sa généricité (de quel module il provient).

Cette activité de conception logicielle dans Mozilla génère ainsi les artefacts suivants : document de conception logicielle, document de traçabilité, document de requêtes de changements [104], document de cas de tests [15].

5.2.1.4 Construction du logiciel

L'activité de construction du logiciel représente la plus importante activité du cycle de vie de logiciel dans les communautés de F/OSS [98], car c'est au cours de cette activité que le logiciel prendra forme et aboutira après vérification/révision à une

première version stable du système. Le principe de base du développement de F/OSS réside dans le partage du code source. Cela permet aux développeurs de coopérer sous un modèle de revue systématique par les pairs et d'en tirer l'avantage du débogage en parallèle, qui mène à des avancements rapides et à des innovations [79]. Dans chacun des F/OSS retenus, il existe des lignes directives (*guidelines*) de développement qui incluent les styles ou standards de codage [98]. Certains de ces projets (par exemple GNOME [60]) possèdent une équipe d'assurance qualité qui effectue les contrôles et qui s'assure que les processus et *guidelines* sont respectés ou suivis [98]. Le fait que les développeurs ne travaillent que sur des tâches sur lesquelles ils ont une réelle passion [102] et qui sont en rapport avec leur expertise, contribue à avoir un code écrit avec plus de précision et de créativité [94]. Les équipes de développement et de tests de projets libres comprennent des individus qui ne se rencontrent jamais face à face, ni même par téléphone 94], de ce fait, toute l'information sur les F/OSS est donc enregistrée sous forme électronique. Ce qui contribue grandement à la transparence de tout document et de tout processus dans les F/OSS.

Étant donné que les projets Apache, GNOME, Linux et Mozilla, utilisent la modularité du code comme pratique de qualité [26, 36, 60, 94, 104, 105], précisément lors de la conception [98], chaque module possède sa propre liste des fonctionnalités à réaliser, disponible sur le site web de la communauté. Au cours de l'activité de construction du logiciel, les développeurs mettent tout leur savoir faire. Ils sont généralement passionnés et expriment leur créativité dans le code selon le style de codage prescrit dans le *guideline* de leur communauté de F/OSS. Ceux ayant fournis des contributions significatives, gagnent le respect des membres de la communauté et se voient attribuer plus de responsabilités. Le choix des tâches à réaliser est fait selon la classification courante des tâches spécifiques du projet à réaliser dans chacun des modules, mises sur le site web de la communauté [60, 94, 107]. D'un autre côté, les développeurs peuvent mettre à jour une fonctionnalité du logiciel en se basant sur les

travaux à réaliser ayant été publiés sur le site web de la communauté ou par l'entremise des *mailing lists*. Nous rencontrons dès lors de multiples développeurs répartis à travers le monde, pouvant travailler sur la même tâche. Ce qui produit différentes versions de la même contribution et augmente la probabilité d'avoir des contributions importantes des fonctionnalités du logiciel. L'aspect distribué des projets de logiciels libres, mène à une qualité élevée de logiciel, car de nombreux développeurs, qui sont également des utilisateurs du logiciel, participent activement et de manière itérative à la détection et à la correction des défauts du logiciel. Il convient donc à ce niveau de se poser les questions suivantes: comment se fait la communication dans les projets libres? Comment sont sélectionnées les bonnes contributions et corrections qui doivent être intégrées dans le référentiel de documents du projet et pour finir, quelle est l'autorité habiletée à jouer ce rôle? En d'autres termes, qui le fait?

Dans Apache, selon Mockus, Fielding et Herbsleb [94], ce sont les développeurs de base qui décident de travailler sur un problème, de même qu'ils décident de la solution retenue pour ce problème. Avant de trouver la solution à un problème, ils coordonnent ainsi leurs efforts [52] et ont recours à une prise de décision concernant le choix des différentes possibilités pour déterminer la solution appropriée. Des utilisateurs peuvent fournir une solution qui fonctionne, cependant, même lorsque celle-ci fonctionne, elle peut comporter des caractéristiques non souhaitables en tant que solution générale et quelque fois cette solution peut ne pas être portable sur d'autres plateformes. Ces raisons appuient leur choix concernant une solution. Dans le cas où plusieurs solutions de rechange existent, les développeurs envoient des alternatives sur la liste de diffusion dans le but d'obtenir les feedbacks du reste du groupe avant d'en élaborer une nouvelle. Selon Yutaka Yamauchi et ses collaborateurs [123], le processus de F/OSS est résumé comme : « une action, l'expérimentation et l'innovation suivies par la coordination, les revues par les pairs, l'amélioration et la stabilité ». Cependant, d'autres études ont démontré le contraire,

c'est le cas du projet Apache dans lequel la coordination apparait en premier lieu. Par exemple, lorsque l'équipe de base trouve des solutions alternatives à un problème donné, elle a invité une revue pour l'obtention des feedbacks via les *mailing lists* des développeurs et a ensuite déterminé une solution avant qu'elle ne soit développée [94]. En d'autres termes, elle a lancé ou initié une revue du code de la solution par les pairs afin d'obtenir des feedbacks usuels. Elle publie cette information sur les *mailing lists* des développeurs pour les en informer, pour qu'ils puissent vérifier et corriger les défauts, et ainsi contribuer par des feedbacks. À la fin de ce processus, une solution a été identifiée et un rapport de la revue est généré. Dans GNOME [60], ce sont les mainteneurs du projet pour chaque module qui acceptent ou rejettent une contribution de solution soumise.

Lorsqu'une solution est identifiée, chaque développeur fait des changements sur sa copie locale du code source et les teste sur son serveur. Ce niveau de tests s'apparente aux tests unitaires et dans certains cas à des tests fonctionnels dans le développement commercial [94]. A l'opposé de Mozilla qui pratique en plus des tests de régression [104], il n'existe aucun autre processus de tests dans Apache. Après les tests unitaires, le développeur de base peut soit soumettre le changement directement (par exemple, cas d'une revue de type CTR dans Apache [105] – confer chapitre 3) ou poster le *pacth* (par exemple, cas d'une revue de type RTC dans Apache [105] – voir chapitre 3) sur la *mailing list* pour une revue par les pairs avant la soumission finale. Les requêtes de changements peuvent également être postées sur le système de suivi de bogues, généralement Bugzilla dans les F/OSS retenus [25, 67], cependant celles postées sur la liste de diffusion sont les plus prioritaires [94]. Certains systèmes (par exemple, cas de Linux) ont recours à des branches [28, 60, 91] pour différencier les changements réalisés pour chaque version du système. Par contre d'autres systèmes, cas d'Apache [94], nécessitent que les programmeurs décrivent les changements effectués pour chacune des transactions vers le système de contrôle de versions, afin de différencier les versions. Habituellement dans les projets retenus, tout changement

dans le code d'une solution pour une version stable du système, requiert d'être révisé avant d'être intégré dans le CVS [28, 94]. Tant dans Apache [94] que dans Linux [28], Mozilla [94, 104] et GNOME [60], compte tenu du fait que n'importe qui peut souscrire à une liste de diffusion, les changements sont donc révisés par une multitude de personnes externes au groupe de développement de base. De cela, il résulte de nombreux feedbacks usuels avant que le logiciel soit formellement livré sous forme de paquet (*package* en anglais, représentant une version stable du système). Comme ces communautés utilisent des CVS afin de contrôler les différentes versions de logiciels, il est donc possible de retrouver l'historique des changements ayant été opérés et d'en constituer un document de changements. Il est également possible de garder la trace du statut du projet [28, 60, 94], ainsi que du type de résolution indiquant comment les bogues ont été résolus [28].

De l'observation du déroulement de l'activité de construction du logiciel dans les communautés de F/OSS, nous avons constaté les aspects suivants :

- Très peu de documentation des étapes du processus de construction de logiciel existe dans les projets de F/OSS retenus pour notre étude. Généralement, elle est même inexistante. Cela dit, les documents décrits par la norme ISO/IEC 29110 sont bel et bien utilisés dans les projets de logiciel libres et open source [67] et peuvent être reconstitués à partir des archives de courriels, des documents présents dans les *bugtracking* (systèmes de suivis de bogues) et CVS des différents projets logiciels libres. Nous citons : « *bug report, traceability record* ».
- Des cas de tests unitaires sont appliqués dans les projets de F/OSS retenus afin de vérifier le parfait fonctionnement d'un composant du logiciel, ainsi que de corriger les défauts du logiciel. De plus, nous notons que des tests fonctionnels sont appliqués et menés non seulement par de multiples programmeurs et les développeurs de base répartis à travers le monde et utilisant l'Internet comme moyen de communication, mais aussi par les autres

participants et utilisateurs du logiciel [102]. Ceci est un processus permettant de corriger rapidement les défauts, lequel augmente de manière considérable la qualité du logiciel dans les projets de F/OSS. Ce processus se fait de manière itérative dans chacun des modules du logiciel. De plus, certaines des communautés retenues, ici, Mozilla [94, 104], GNOME [60, 67] et Linux [67], appliquent des tests automatisés dans leur développement. Nous notons également la présence des tests de régression : cas Mozilla [104].

- De notre constat, les rôles impliqués dans cette activité sont ceux de programmeurs et de chef de projet (ce dernier qui n'est rien d'autre que leader technique. Cependant, l'appellation diffère d'une communauté à une autre. Dans Eclipse, ils portent le nom de *committers*; dans Apache, ce sont *les programmeurs de base* (AG) ou *committers*; dans GNOME, les *mainteneurs*), ce qui est conforme à la norme.
- **Cas de Apache (Apache Software Foundation)**: chaque membre du groupe de base d'Apache peut voter sur l'inclusion des changements de code et possède un accès en soumission (soumission de code ou de documents) au CVS s'il le désire [94]. Le groupe de base d'Apache est généralement constitué de 25 personnes, cependant seules 15 personnes sont actives et sont connues sous l'appellation développeur de base. Pour des changements majeurs du code qui affecteront les autres développeurs, des votes sont requis : 3 votes positifs et aucun vote négatif sont requis pour accepter un changement de code, ce qui limite donc le nombre de participants à la revue du code [52]. Il n'existe pas un processus unique de développement, et donc, chaque développeur de base fait des itérations à travers différentes actions tant qu'il travaille sur le code source du logiciel [94]. Ces actions sont de découvrir qu'un problème existe ou qu'une nouvelle fonctionnalité est nécessaire, de déterminer si oui ou non un volontaire travaillera sur la question, d'identifier la solution, de développer et tester le code dans sa copie locale du code source, de présenter les changements de code à AG pour

révision et enfin de soumettre le code et la documentation au dépôt de documents du projet. Dépendamment du changement ou de la correction à faire, ce processus peut impliquer plusieurs itérations avant d'aboutir à une conclusion.

- **Cas de Linux** : particulièrement, pour le compilateur **gcc** (GNU Compiler Collection), l'outil DejaGnu [67] est utilisé afin d'automatiser les tests d'autres programmes, ceci dans le but de maintenir un niveau élevé de qualité tout au long de la vie d'un logiciel.
- **Cas de GNOME** : utilise Autoconf [67] pour faire les constructions et tests automatiques du logiciel.
- **Cas de Mozilla** : 22 leaders de projets [94]. Tinderbox est utilisé pour faire des constructions automatiques du logiciel [67, 94, 104].

En moyenne, pour qu'un projet marche, ou ait du succès, il faut en moyenne au moins 15 programmeurs fixes [26] qui joueront dans la plupart des cas, les rôles de leaders techniques (ce sont les programmeurs de base, ayant le droit d'accéder au référentiel de données du logiciel). De plus, ce sont les gestionnaires de projets, assurant la gestion et la supervision technique des projets développés dans les communautés de logiciels libres.

5.2.1.5 Intégration et tests

L'activité de tests est la seconde activité la plus importante du modèle de développement de F/OSS [98]. Dans un projet logiciel libre, l'intégration consiste habituellement: « à écrire quelques pages de *man*[25], à s'assurer que le logiciel se construit sur tous les systèmes auxquels le développeur a accès, à nettoyer le fichier « Makefile » pour le débarrasser de toutes les commandes superflues qui se sont

[25] Ce sont des pages d'aide en ligne.

accumulées pendant la phase d'implémentation, à écrire un LISEZMOI, à faire une archive, à la télécharger sur un site ftp anonyme quelque part et à poster une annonce dans la liste de diffusion ou le forum de discussion des utilisateurs concernés » [35].

En 2008, Tobias Otte et ses collaborateurs [98] mènent une étude montrant que plus de la moitié des projets suivent une approche structurée de tests. Dans cette même étude, il est mentionné que la majorité des défauts, généralement les défauts majeurs ou bogues difficiles, sont trouvés par les utilisateurs du logiciel, qui sont eux-mêmes des développeurs. Ce résultat est soutenu par d'autres auteurs [102, 104, 113]. De plus, il est montré que 46% des projets de F/OSS font des corrections de bogues avant que le code ne soit soumis au référentiel de documents. Ce résultat soutient clairement l'idée de Fielding et ses collaborateurs [94], ainsi que celle de Mohamed et Swapna [28]. Cette même étude montre que la grande majorité des projets de F/OSS garde trace majoritairement des défauts de code et moyennement de trace de suivi des problèmes dans les exigences, la conception et la documentation. De même qu'il y a une excellente communication entre les programmeurs de projets de F/OSS et les utilisateurs, se traduisant par beaucoup de *feedbacks*.

Étant donné que les F/OSS retenus sont divisés en différents modules qui sont maintenus chacun par un ou plusieurs développeurs dépendamment de la communauté de F/OSS. L'intégration du projet de logiciel libre dans son ensemble débutera donc de manière intuitive par la coordination et l'intégration des différentes contributions des autres développeurs de chaque module du projet, suivi de la coordination et l'intégration des différents modules. Les tâches de coordination et d'intégration de toutes les contributions des développeurs d'un modules sont faites par les chefs qui sont les mainteneurs ou gestionnaires de ces modules [59, 94, 104] et de manière automatique dans certains cas (par exemple dans Linux [49], Mozilla [104]). Habituellement, chaque module de projet possède sa propre équipe

d'assurance qualité [60, 104], portant le nom d'*équipe de release* dans GNOME [60], exception faite cependant du projet Apache, qui lui est maintenu et géré par les développeurs de base [94]. Généralement dans les F/OSS, particulièrement dans Linux, Apache, Mozilla et GNOME, pour un module donné, une fois qu'une solution a été identifiée et que les différents changements ont étés effectués et testés par les différents participants au projet, la solution retenue doit être intégrée au référentiel de composants logiciels. Comme déjà mentionnée ci-dessus, la solution retenue est révisée par les pairs avant d'être soumise au référentiel de composants logiciels. Lorsque le projet se rapproche de la date planifiée pour la *release*[26], l'un des gestionnaires techniques du projet[27] de F/OSS se porte volontaire d'être gestionnaire de version. En rappel, les gestionnaires techniques dans Apache, sont les développeurs de base [94], dans Linux et GNOME, ils sont connus sous l'appellation de mainteneurs de projet [40, 60], dans Mozilla, ce sont les membres de l'organisation Mozilla, précisément les propriétaires de modules [104]. Le fonctionnement diffère cependant d'une communauté à une autre. Par exemple :

- Cas Apache : le gestionnaire de version est responsable d'identifier les problèmes critiques (le cas échéant) prévenant la livraison, de déterminer quand est-ce que ces problèmes ont été réparés et que le logiciel a atteint un point stable, et pour finir, de contrôler l'accès au référentiel de documents afin que les développeurs ne puissent pas modifier des choses qui ne devraient pas être changées avant la version à poster sur le site web [94]. Apache fonctionne de la même façon que Linux [28, 102].

« The role of release manager is rotated among the core developers with the most experience with the project. » [94].

---

[26] La date de release est considérée dans les F/OSS comme la date de révision du produit avant de poster la version stable retenue du système sur le portail web de sa communauté de F/OSS.

[27] Ce sont des développeurs de base ou de confiance de projets libres qui se sont vue assigner d'autres responsabilités. Ils ont le droit d'accéder au référentiel de documents du projet.

- Cas Mozilla: le projet utilise un processus continu et quotidien de construction du système grâce à l'outil Tinderbox, permettant de montrer les parties du code qui ont des problèmes pour certaines constructions et sous certaines plateformes [94]. De plus, dans Mozilla, les « *smoke tests* » sont exécutés pour s'assurer de la bonne marche de la construction du système faite [94].

Prenons le cas de Debian. Cette distribution Linux est l'une des plus vieilles distributions GNU/Linux existantes et l'un des systèmes complexes qui est construit à partir de composants logiciels libres [113, 119]. Tout comme dans GNOME, Debian possède un groupe de mainteneurs de projets. Ils ont le droit d'accéder au référentiel de composants logiciels et jouent le rôle de leaders techniques. L'infrastructure en place permet de superviser, de coordonner et de maintenir les projets de F/OSS. Elle est constituée des outils F/OSS: CVS [67], debbugs [10, 64].

Les archives du référentiel/dépôt de documents et logiciels de Debian, contiennent les informations sur le nom, la version, le mainteneur, la licence, la taille et autres informations sur le *package*, avec le plus important étant que, chaque *package* contient des références[28] pour d'autres *packages* nécessaires à l'exécution [113]. Ainsi, il devient donc facile et rapide d'intégrer tous les composants logiciels, en ne considérant que les informations de dépendances de type « requis », et ce, de manière intuitive. Les mainteneurs de projets définissent des cas de tests que les développeurs exécutent. Les utilisateurs participent aux tests du système et découvrent la majorité des défauts importants ou majeurs et proposent des solutions [113]. Les utilisateurs contribuent aux projets, par des *feedbacks* aux programmeurs sous forme de rapports de bogues, qui constitueront en plus des défauts détectés par les développeurs, de document complet de rapport de test. Cette façon de fonctionner est courante dans les

---

[28] Les références dont il est questions consistent en des informations de dépendances portant les statuts suivants : « requis », « recommandé » ou « suggéré » [113].

projets de F/OSS, en l'occurrence Apache, Mozilla. Actuellement, la majorité des bogues est soit résolue par des corrections du code source du logiciel, soit par un *patch* soumis aux autres développeurs sur la *mailing list* pour l'amélioration de la fiabilité du système [28].

De plus, un bon environnement intégré de composants logiciels dans les F/OSS facilite la réutilisation [113]. Dans Linux, l'un des avantages significatif provient des **efforts continuels** d'amélioration de la fiabilité qui sont reflétés dans **les lots de correction par mois** ou **deux fois par semaine**, entrepris par les développeurs de la communauté [28], ceci à l'opposé des produits commerciaux, où les bogues sont corrigés une fois qu'ils ont été rapportés par les clients.

Debian est l'un des exemples marquant de systèmes complexes construits avec des composants F/OSS [40, 113]. Puisque les communautés de F/OSS pratiquent le libre accès au code source et la modularité du code, ces composants logiciels libres peuvent donc être réutilisés afin de faciliter la construction d'un nouveau composant. La réutilisation du code de composants logiciels préalables est une pratique très utilisée et fréquente en logiciels libres [28, 104, 113].

5.2.1.6 Livraison du produit

Nous n'avons pas trouvé une documentation nous expliquant comment se fait réellement l'activité de livraison de produits logiciels libres. L'extraction des pratiques utilisées dans le processus de livraison du produit dans les F/OSS a été plutôt difficile. Nous nous sommes ainsi servis de nombreux travaux menés en logiciels libres, ainsi que des informations relatives aux projets libres retenus se trouvant sur leur site web respectif, afin de déduire le processus de livraison de logiciel.

La livraison du produit en logiciels libres est une tâche plutôt ardue car elle nécessite de passer à travers un processus de gestion de versions. Erenkrantz [49] définit le processus de gestion de versions en logiciels libres comme étant : « une combinaison complexe de sous-processus et des outils choisis pour soutenir les objectifs et les propriétés spécifiques du projet ».

De façon très abstraite, nous savons que le logiciel est testé rigoureusement à la fois par les utilisateurs et les développeurs, jusqu'à obtention d'une version stable du logiciel. Eric Raymond [102] qualifie cette façon dont les logiciels libres sont testés par les participants aux projets, de test par plusieurs « *eyeballs* », en rapport avec le projet Linux. Une fois le logiciel testé et que la date planifiée pour la livraison du produit se rapproche, les leaders techniques ou coordonateurs de chaque projet, procèdent à la gestion des versions [49]. Quelques fois, les programmeurs peuvent rédiger la documentation de maintenance, tâche qui n'est cependant pas obligatoire dans le monde du libre. Si un tel document existe, il est très souvent sous-forme de page d'aide en ligne, et tant les développeurs que les coordonateurs ou mainteneurs de chaque projet, se chargent de vérifier la documentation de maintenance jusqu'à sa consistance. Une fois la consistance de la documentation de maintenance atteinte, l'un des coordonateurs de projet peut se charger de la stocker dans la pile de documents *baselined*, via le CVS, et parallèlement, il met à jour cette documentation en ligne. Le coordonnateur de projet joue le rôle de gestionnaire de versions.

Tel dans Apache que dans Linux, le rôle de gestionnaire de versions circule entre les différents mainteneurs ou chef du projet [49, 94]. Cependant, à l'opposé de Linux où le gestionnaire de versions est désigné par les autres membres de mainteneurs de projet [49], dans Apache, l'un des membres des développeurs de base se porte volontaire pour assumer le rôle de gestionnaire de versions [94]. D'autres communautés par contre peuvent avoir un ou plusieurs gestionnaires de versions, cas : GNOME [60] et Mozilla [94]. L'hiérarchie voudrait que, les participants

externes au groupe de développeurs de base du projet ou mainteneurs du projet, doivent soumettre les *patchs* aux mainteneurs de projets et non au gestionnaire de versions et les mainteneurs de projets peuvent soumettre des *patchs* aux gestionnaires de versions pour intégration [49]. Cette intégration des différents *patchs* se fait de façon automatique dans certains projets libres, par exemple, Linux [49] et Mozilla [104]. Habituellement dans Linux [49, 74] et GNOME [60], chaque version du logiciel porte un numéro pour son identification, consistant en des entiers, représentés sous le format *x.y.z.* Le gestionnaire de versions est l'autorité responsable qui décide des *patchs* et des changements, qui doivent être inclus dans une version, de même qu'il est responsable de créer les différents artefacts de la livraison du produit [49].

Le gestionnaire de versions annonce ainsi une révision candidate du logiciel sur la *mailing list* des développeurs, question d'obtenir des feedbacks préliminaires [49, 94]. Dans Apache, la révision candidate énoncée sur la *mailing list* par le gestionnaire de versions, est destinée à des développeurs spécifiques [49, 105]. Le gestionnaire de versions peut exécuter des tests de régression [49, 104], afin de compléter les tests unitaires et fonctionnels ayant été effectués préalablement. Lorsqu'un problème est résolu, un programmeur peut ainsi décider d'intégrer le test de régression contrôlé dans le dépôt de documents, tâche qui n'est cependant pas obligatoire. Une fois la révision candidate effectuée, le code du produit est publié sur le site web de la communauté de F/OSS pour révision proprement dite pour une certaine durée, en vue de recevoir des feedbacks des utilisateurs [49]. Ce processus permet de stabiliser le produit et ainsi le livrer une fois stabilisé. La plupart des F/OSS note le gel du code, période pendant laquelle le gestionnaire de versions peut rejeter tout changement ou autres corrections de bogues [96]. Une fois les changements stabilisés, le gestionnaire de versions met à jour le site web de la communauté par la distribution ou livraison sur leur site web d'une version emballée de ce logiciel de façon propice à attirer des utilisateurs non-techniques. La distribution d'un produit ou d'un document dans les F/OSS consiste en deux principaux aspects : la **visibilité** et l'**accessibilité** [49]. Le

format de tous les fichiers du produit logiciel pour la version stable du système, distribuée sur le site web du projet F/OSS, est généralement le gzip [49]. Nous notons cependant que le processus de gestion des changements diffère d'une communauté à une autre. Par exemple : Mozilla exécute un processus de construction continue du logiciel grâce à l'outil Tinderbox [94], quant à Apache, l'un des chefs du projet se porte volontaire pour être le gestionnaire de versions [94].

CHAPITRE VI

NORME ISO/IEC 29110 ET COMPARAISON DES PROCESSUS

6.1 PRÉSENTATION DE LA NORME ISO/IEC 29110

6.1.1 Définition

Nous commençons en entrée de jeu par la définition du **génie logiciel** car la norme ISO/IEC 29110 est une norme d'ingénierie. Selon *IEEE Computer Society* [27], le génie logiciel est: « *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software* ».

La norme ISO/IEC 29110 [73] est une norme ayant été développée par l'organisme : ISO (Organisation Internationale de Standardisation), du sous-groupe IEC (Commission Électrotechnique Internationale), afin d'améliorer le produit et/ou la qualité du service et la performance des processus. Cette norme tire profit de certaines pratiques des normes ISO/IEC 12207[29] et ISO/IEC 15289[30]. La norme

---

[29] Norme ISO, du sous groupe IEC, conçue pour le processus de manière générale du cycle de vie du logiciel. Selon Wikipedia [11], elle a pour objectif de définir toutes les tâches requises pour développer et maintenir un logiciel.

ISO/IEC 29110, précisément de son appellation complète ISO/IEC PDTR 29110-5-1.3, décrit le profil de cycle de vie pour les très petites organisations (TPO)[31]. En d'autres termes, elle fournit une gestion d'implémentation et un guide d'ingénierie pour un profil de TPO à travers la gestion de projet et les processus d'implémentation de logiciel.

6.1.2 Guide d'ingénierie et gestion du profil basique d'une VSE

Selon la norme [73], ce guide assure la gestion de projets et les processus d'implémentation de logiciels qui intègrent des pratiques basées sur la sélection ISO/IEC 12207- *Ingénierie logicielle et systèmes – processus du cycle de vie de logiciel* et ISO/IEC 15289 – *Ingénierie logicielle - Processus du cycle de vie de logiciel – lignes directives pour la documentation du cycle de vie de logiciel* (voir figure 6.1 pour la description du guide).

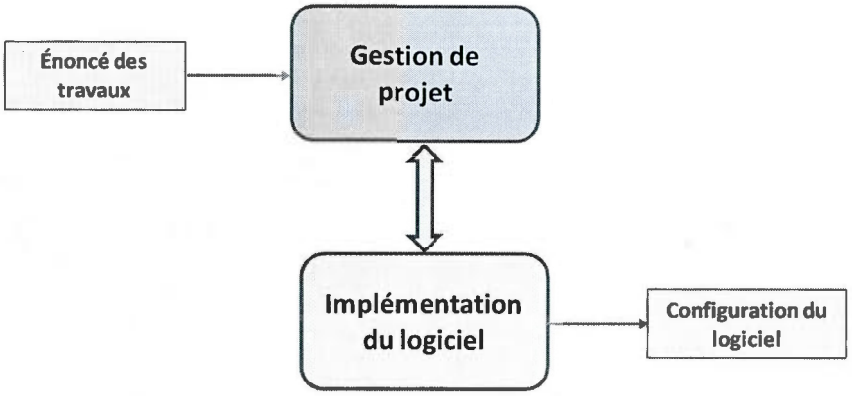

Figure 6.1 Processus du guide du profil de base traduit de [73]

[30] Elle a été conçue comme support ou documentation à la norme ISO/IEC 12207.

[31] Selon ISO/IEC 29110 [73], les très petites organisations (en anglais : *Very Small Entities* (VSE)), sont des entreprises, organisations, départements ou projets, constitués d'**au maximum 25 personnes**, dédiées au développement de logiciel.

Le guide d'ingénierie et gestion du profil de base d'une VSE est constitué de deux principaux aspects :

- Le processus de gestion de projet;
- Le processus d'implémentation de logiciel.

L'usage de ce guide par les VSEs, leur permet de bénéficier des aspects suivants :

- Un ensemble convenu des exigences du projet et des produits attendus, est livré au client ;
- Un processus discipliné de gestion est effectué. Il fournit la visibilité sur le projet et les actions correctives des problèmes et déviations du projet;
- Un processus d'implémentation logicielle est suivi. Il satisfait aux besoins du client et s'assure de la qualité des produits.

6.1.2.1 Processus de gestion de projet

Le processus de gestion de projet (voir figure 6.2) est un processus permettant d'établir et de mener à bien de façon systématique, les tâches du projet de réalisation de logiciels, ce qui permet de répondre aux objectifs du projet dans la qualité attendue, le délai et les coûts [73].

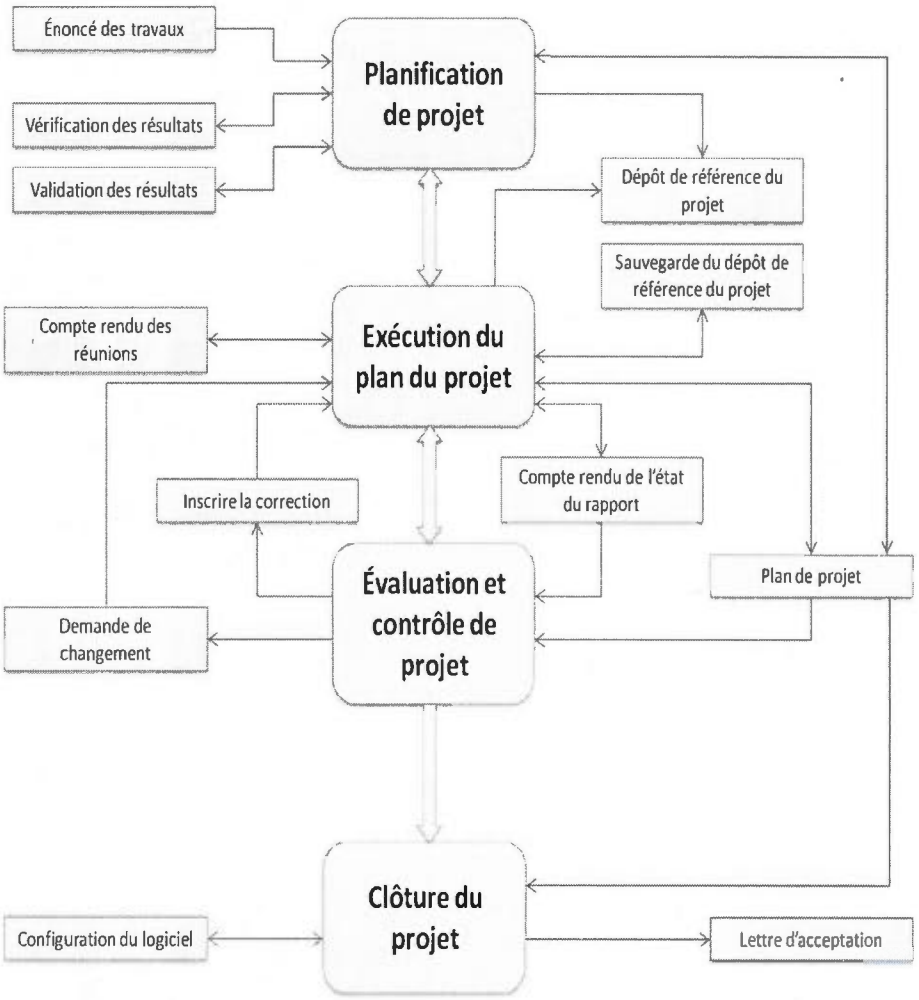

Figure 6.2 Diagramme du processus de gestion de projet traduit de [73]

Ce processus de gestion de projet est constitué des activités ou processus suivants :

- **PM.1 Planification du projet :**

Elle documente les détails de planification nécessaires pour gérer le projet. Cette activité a pour objectifs de :

- Réviser l'état du travail ainsi que les tâches nécessaires pour fournir un contrat des livrables et pour satisfaire aux exigences du client ;

- Réviser le cycle de vie du projet en incluant les dépendances des tâches et leur durée ;
- Définir la stratégie d'assurance qualité du projet à travers les vérifications et validations des livrables/produits du travail, effectuées au cours des revues par l'équipe de travail et le client ;
- Identifier et définir les rôles et les responsabilités du client et de l'équipe de travail ;
- Fournir les ressources du projet et les besoins en formation ;
- Estimer l'effort, le coût et le calendrier ;
- Identifier les risques du projet ;
- Définir le contrôle de versions du projet et la stratégie de *baseline* ;
- Définir le dépôt du projet pour sauvegarder, gérer et livrer les produits contrôlés et les versions des documents et les *baselines* (base de référence).

➢ **PM.2 Exécution du plan de projet :**

Cette activité implémente la documentation du plan de projet. Elle a pour objectifs :

- Aboutir à un commun accord entre le gestionnaire de projet (PM) et leader technique (TL) sur l'affectation des tâches;
- Mettre à jour le *Rapport de l'état d'avancement du projet*;
- Analyser et évaluer les requêtes de changements pour le plan par rapport au coût, au calendrier et aux exigences techniques;
- Faire l'approbation des changements sur le plan;
- Mener des révisions et des accords à la fois par l'équipe de travail (WT) et le client (CUS);
- Sauvegarder le dépôt de documents du projet (*Projet repository*) et sa récupération si nécessaire.

➢ **PM.3 Contrôle et évaluation du projet :**

Cette activité surveille et évalue la performance/rendement du plan par rapport aux soumissions documentées. Elle a pour objectifs de :

- Réviser le rendement et le progrès du plan par rapport aux cibles;
- Identifier et évaluer les coûts importants, les calendriers et les écarts de performance technique et les problèmes;
- Examiner les risques du projet et identifier de nouveaux risques;
- Documenter les requêtes de changements, les actions correctives définies et le suivi des changements pour finir.

➢ **PM.4 Fermeture du projet :**

Elle fournit la documentation et les produits du projet selon les exigences du contrat. Elle a pour objectifs de :

- Livrer le produit comme spécifié dans les instructions de livraison;
- Appuyer l'acceptation du produit au client selon les instructions de livraison;
- Achever le produit et faire la signature du rapport d'acceptation.

6.1.2.2 Processus d'implémentation de logiciel

Le processus d'implémentation de logiciel (voir figure 6.3) est la performance systématique des activités d'analyse, de conception, de construction, d'intégration et de tests pour de nouveaux produits logiciels ou de produits logiciels modifiés selon les exigences spécifiées [73].

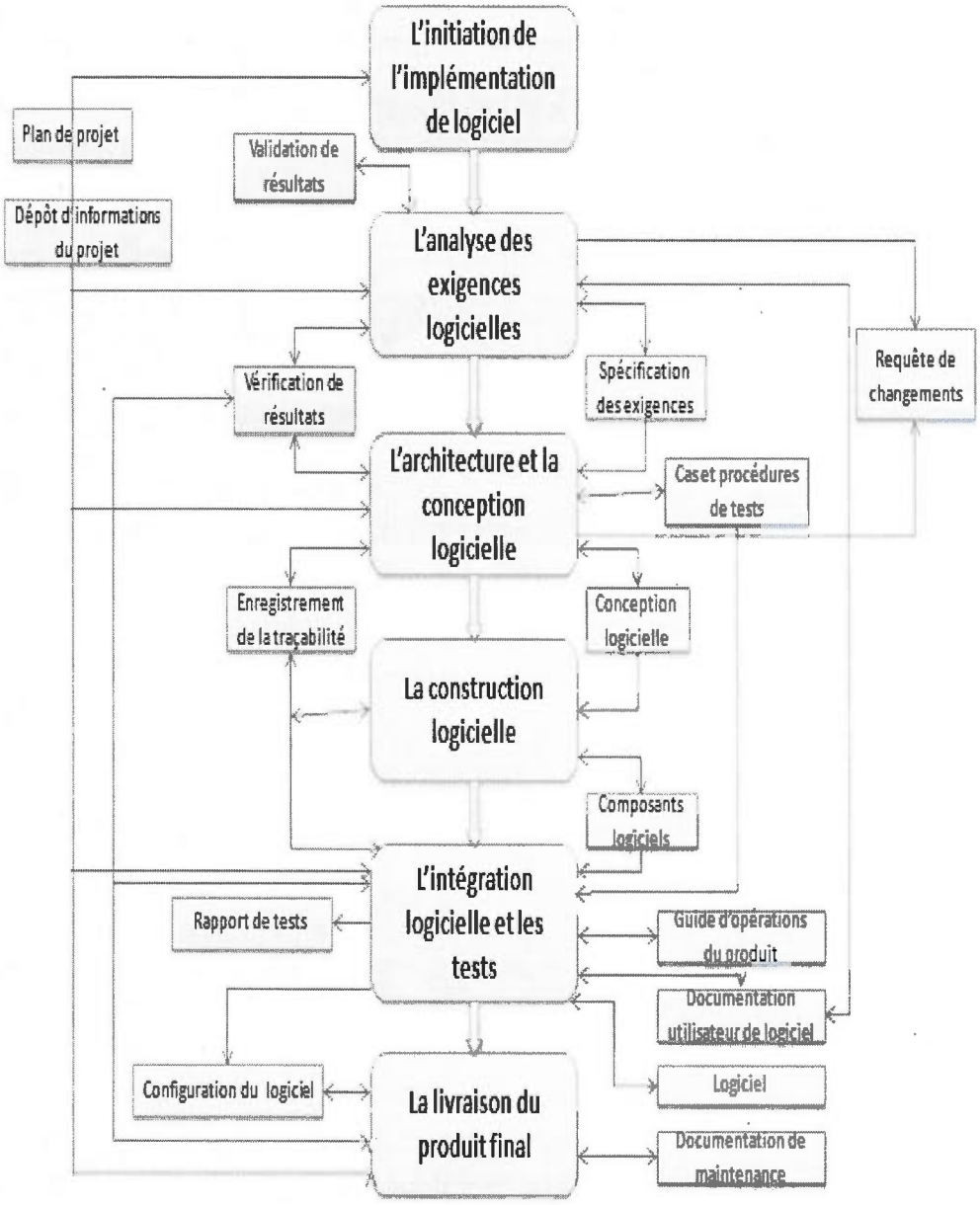

Figure 6.3 Diagramme d'implémentation de processus traduit de [73]

L'implémentation de logiciel (en anglais *Software Implemention*) est constituée des activités ci-dessous :

➢ **SI.1 Initiation d'implémentation du logiciel :**

L'activité d'initiation d'implémentation du logiciel selon la norme ISO/IEC 29110 assure que le plan de projet établi soit accepté par l'équipe de travail, le leader technique et le gestionnaire de projet. Cette activité a pour objectifs de :

- Réviser le plan de projet par l'équipe de travail afin de déterminer l'affectation des tâches;
- Soumettre le plan de projet par l'équipe de travail et le gestionnaire de projet;
- Établir un environnement de développement.

➢ **SI.2 Analyse des exigences du logiciel :**

Cette activité a pour but de s'assurer que les spécifications du projet soient validées par le client, conformément à ses besoins [73]. Cette activité a pour objectifs de :

- Faire l'élicitation, l'analyse et la spécification des exigences du client;
- Avoir un accord sur les exigences du client;
- Faire la vérification et la validation des exigences;
- Contrôler la version des produits d'exigences de logiciel.

➢ **SI.3 Architecture et conception :**

Le but de cette activité est de transformer les spécifications du logiciel en une architecture logicielle de système et une conception détaillée du logiciel [73]. Cette activité a pour objectifs de :

- Concevoir l'architecture logicielle, les composants logiciels et les interfaces associées;
- Faire la conception détaillée des composants logiciels et des interfaces;
- Faire la révision des spécifications d'exigences logicielles par l'équipe de travail;
- Vérifier les cas et les procédures pour les tests d'intégration;
- Faire la traçabilité des exigences logicielles pour la conception logicielle, les cas de tests et les procédures de tests;

- Faire la conception de produits et documents sous le contrôle de version.

➢ **SI.4 Construction logicielle :**

Cette activité a pour but de développer le code du logiciel et le modèle de données en se basant sur la conception du logiciel [73], faite lors de l'activité d'architecture et de conception. La construction logicielle a pour objectifs de :

- Examiner la conception logicielle par l'équipe de travail afin de déterminer l'affectation des tâches et la séquence de construction;
- Coder les composants logiciels et appliquer les tests unitaires;
- Faire la traçabilité entre les composants logiciels et la conception logicielle.

➢ **SI.5 Intégration et tests :**

Cette activité s'assure que les composants logiciels intégrés satisfont les spécifications du logiciel [73]. Elle a pour objectifs de :

- Faire la révision du plan de projet par l'équipe de travail en vue de déterminer l'affectation des tâches;
- Comprendre les cas de tests, les procédures de tests et l'environnement d'intégration;
- Intégrer les composants logiciels, corriger les défauts et documenter les résultats;
- Faire la traçabilité des exigences et conception au produit logiciel intégré;
- Vérifier et documenter la documentation utilisateur et opérationnelle du logiciel;
- Vérifier la base de référence de logiciel (*Software Baseline*).

➢ **SI.6 Livraison du produit :**

L'activité de livraison de produit consiste à fournir au client un produit qui respecte ses besoins, et qui soit fiable [73]. Elle a pour objectifs de :

- Vérifier la documentation de maintenance;

- Faire la livraison du produit logiciel et la documentation applicable en concordance avec les instructions de livraison.

6.2 COMPARAISON DES PROCESSUS

Nos approches de comparaison sont divisées en deux exercices :

- **Premier exercice :** L'évaluation du processus de développement de logiciel libre, en vue de déterminer les différences entre les processus libres et ceux en génie logiciel;
- **Second exercice :** L'analyse de compatibilité des activités de la norme ISO/IEC 29110 avec les F/OSS.

6.2.1 Différences entre le processus de développement de logiciels libres et le développement classique de logiciels

Dans cette section, nous comparons les processus de développement libre avec les processus de développement de logiciels en génie logiciel en se basant sur l'approche d'analyse décrite au chapitre 4 – section 4.4.1. La norme ISO/IEC 29110 est utilisée comme cadre de référence pour supporter notre comparaison.

Il faudrait tout de même garder à l'esprit les méthodologies classiques[32] de développement de logiciels (confer - ANNEXE C pour les détails concernant certaines méthodologies de développement de logiciels) : le modèle en cascade, le modèle en V; car elles sont celles utilisées le plus souvent en entreprise (car très structurées), respectivement pour des projets de petite taille et des projets de taille

[32] Elles font référence aux cycles séquentiels de développement, citons : le modèle en V, le modèle en cascade, le modèle par incréments. Voir ANNEXE C pour les détails.

moyenne. Quant au modèle par incréments, des recherches ont montré qu'il est utilisé pour le développement des projets de F/OSS [94, 96].

6.2.1.1 Évaluation du processus de développement de logiciels libres

L'évaluation du processus actuel de développement de logiciels libres est réalisée dans cette section conformément aux activités du processus d'implémentation de logiciel présenté dans la norme ISO/IEC 29110 – section 4.3 [73], en vue de relever les différences entre le développement de logiciels libres et le développement classique de logiciels. Rappel chapitre 4 – sections 4.3 à 4.4.1 pour les détails de l'approche utilisée à cet effet.

Les tableaux 6.1 à 6.6 représentent respectivement les évaluations des processus libres, précisément : processus d'initiation d'implémentation de logiciel, processus d'analyse des exigences, processus d'architecture et de conception, processus de construction logicielle, processus d'intégration et tests, et pour finir le processus de livraison de logiciel.

Le signe ✓ représente non seulement la façon dont est réalisée une activité d'un processus libre, mais il est également utilisé afin d'identifier les différents rôles et les artefacts associés à cette activité. Un X signifie que la tâche à accomplir est quelque fois réalisée. Un O signifie qu'un document ou un commentaire est optionnel, car la réalisation de cet artefact ou commentaire dans les F/OSS n'est pas obligatoire, compte tenu du bénévolat des participants aux F/OSS. Le client en logiciel libre fait plus référence à l'utilisateur de logiciel, que ce soit un individu ou une entreprise commerciale. Les sigles CUS, AN, DES, PR, TL, PM et WT signifient respectivement : client, analyste, concepteur, programmeur, leader technique, gestionnaire de projet et équipe de travail.

➢ **Activité d'initiation d'implémentation :**

Pratiques définies par ISO/IEC 29110		**Analyse de la réalisation de l'activité et identification des rôles et des artefacts pour chaque activité**											
Référence de la tâche	Description de la tâche	Parfaitement exécutée	Partiellement exécutée	Non exécutée	Aucune connaissance de l'existence de cette tâche	Artefacts ou Commentaires	CUS	AN	DES	PR	TL	PM	WT
SI.1.1	Révision du plan de projet actuel en vue d'aboutir à une compréhension commune.		✓								✓		✓
SI.1.2	Configuration ou mise à jour de l'environnement d'implémentation.	✓									✓		

Tableau 6.1 : Évaluation de l'activité d'initiation d'implémentation de logiciel

➢ **Activité d'analyse des exigences :**

Pratiques définies par ISO/IEC 29110		Analyse de la réalisation de l'activité et identification des rôles et des artefacts pour chaque activité											
Référence de la tâche	Description de la tâche	Parfaitement exécutée	Partiellement exécutée	Non exécutée	Aucune connaissance de l'existence de cette tâche	Artefacts ou Commentaires	CUS	AN	DES	PR	TL	PM	WT
SI.2.1	Affectation des tâches aux membres de l'équipe de travail selon leur rôle, en se basant sur le plan de projet actuel.		✓							✓			✓
SI.2.2	Documentation ou mise à jour de la spécification des exigences.	✓				✓	✓			✓	✓		
SI.2.3	Vérification de la spécification des exigences.					✓				✓	✓		
SI.2.4	Validation de la spécification des exigences.	✓				✓					✓		
SI.2.5	Documenter la version préliminaire de la *documentation uilisateur* du logiciel ou mise à jour du manuel présent. (Optionnel)		(✓, X)			✓				✓			
SI.2.6	Vérification de la *documentation utilisateur* du logiciel.	✓				✓	✓			✓	✓		✓
SI.2.7	Incorporation de *la Spécification des Exigences* et de la **Documentation utilisateur du logiciel* dans la configuration logicielle en *baseline*. *(Optionnel)	✓				✓					✓		

Tableau 6.2 : Évaluation de l'activité d'analyse des exigences

Au niveau de la tâche SI.2.7 de cette évaluation, l'Astérix (*) fait référence au document de *Documentation Utilisateur du logiciel*.

➢ **Activité d'architecture et conception :**

Pratiques définies par ISO/IEC 29110		**Analyse de la réalisation de l'activité et identification des rôles et des artefacts pour chaque activité**											
Référence de la tâche	Description de la tâche	Parfaitement exécutée	Partiellement exécutée	Non exécutée	Aucune connaissance de l'existence de cette tâche	Artefacts ou Commentaires	CUS	AN	DES	PR	TL	PM	WT
SI.3.1	Affectation des tâches aux membres de l'équipe de travail selon leur rôle, en se basant sur le plan de projet actuel.		✓							✓			✓
SI.3.2	Compréhension des spécifications des exigences.	✓				✓				✓	✓		
SI.3.3	Documenter ou mettre à jour la conception logicielle.	✓				✓				✓	✓		
SI.3.4	Vérification de la *conception logicielle*.	✓				✓				✓	✓		✓
SI.3.5	Établir ou mettre à jour les cas de tests et les procédures de tests.	✓				✓					✓		
SI.3.6	Vérification des cas de tests et des procédures de tests.	✓				✓				✓	✓		
SI.3.7	Mettre à jour le *rapport de traçabilité*, en incorporant les cas de tests et les procédures de tests.	(✓, X)				✓					✓		
SI.3.8	Incorporer la *conception logicielle*, les cas de tests, les procédures de tests et le rapport de traçabilité dans la *Configuration logicielle* comme partie du baseline.		✓								✓		

Tableau 6.3 : Évaluation de l'activité d'architecture et de conception

➢ **Activité de construction logicielle :**

Pratiques définies par ISO/IEC 29110		**Analyse de la réalisation de l'activité et identification des rôles et des artefacts pour chaque activité**											
Référence de la tâche	Description de la tâche	Parfaitement exécutée	Partiellement exécutée	Non exécutée	Aucune connaissance de l'existence de cette tâche	Artefacts ou Commentaires	CUS	AN	DES	PR	TL	PM	WT
SI.4.1	Affectation des tâches aux membres de l'équipe de travail selon leur rôle, en se basant sur le plan de projet actuel.		✓							✓			✓
SI.4.2	Compréhension de la *conception logicielle*.				✓								
SI.4.3	Construire ou mettre à jour les composants logiciels basés sur la partie détaillée de la conception logicielle, et définir ou mettre à jour les cas de tests unitaires.	✓				✓				✓			
SI.4.4	Appliquer les cas de tests unitaires pour vérifier que les fonctions marchent parfaitement, selon la conception détaillée du logiciel.	✓			✓					✓			
SI.4.5	Corriger les défauts trouvés jusqu'à atteindre un succès de test unitaire.	✓			✓		✓			✓			✓
SI.4.6	Mettre à jour le *rapport de traçabilité* en incorporant les composants logiciels construits ou modifiés.	(✓, X)				✓				✓			
SI.4.7	Incorporer les composants logiciels et le *rapport de traçabilité* dans la *Configuration logicielle* comme baseline.		✓			✓					✓		

Tableau 6.4 : Évaluation de l'activité de construction logicielle

➢ **Activité d'intégration et tests :**

Pratiques définies par ISO/IEC 29110		**Analyse de la réalisation de l'activité et identification des rôles et des artefacts pour chaque activité**											
Référence de la tâche	Description de la tâche	Parfaitement exécutée	Partiellement exécutée	Non exécutée	Aucune connaissance de l'existence de cette tâche	Artefacts ou Commentaires	CUS	AN	DES	PR	TL	PM	WT
SI.5.1	Affectation des tâches aux membres de l'équipe de travail selon leur rôle, en se basant sur le plan de projet actuel.		✓							✓			✓
SI.5.2	Compréhension des cas de tests et des procédures de tests.	✓								✓			
SI.5.3	Intégration du logiciel en utilisant les composants logiciels et définition ou mise à jour des cas de test et des procédures de tests pour l'intégration.	✓				✓					✓		
SI.5.4	Exécuter les tests en utilisant les cas de tests et les procédures de tests pour l'intégration et documenter les résultats dans le *Rapport de test*.	✓				✓	✓			✓			
SI.5.5	Corriger les défauts trouvés jusqu'à atteindre le succès du test.	✓				✓	✓			✓			
SI.5.6	Mettre à jour le *Rapport de traçabilité* dans le cas échéant.	(✓, X)				O				✓			
SI.5.7	Documenter le *guide d'opération du produit* ou mettre à jour le guide courant, dans le cas échéant.		(✓, X)				✓			✓			
SI.5.8	Vérifier le *guide d'opération du produit*.				✓								

SI.5.9	Documenter la *Documentation utilisateur du logiciel* dans le cas échéant.	(✓, X)					✓			✓			
SI.5.10	Vérifier la *Documentation utilisateur du logiciel*, le cas échéant.	(✓, X)					✓			✓			
SI.5.11	Incorporer le *logiciel*, le *rapport de traçabilité*, le *rapport de test*, le *guide d'opération du produit* et la *documentation utilisateur du logiciel* comme baseline dans la *Configuration logicielle.*		✓								✓		

Tableau 6.5 : Évaluation de l'activité d'intégration et tests

➢ **Activité de livraison de produit :**

En logiciel libre, la tâche *SI.6.3 : Documenter la Documentation de Maintenance ou mettre à jour la courante*, de ce processus est optionnelle. Un programmeur peut décider de réaliser cette tâche. Cependant, dans la majeure partie des cas, elle n'est pas réalisée.

Pratiques définies par ISO/IEC 29110		Analyse de la réalisation de l'activité et identification des rôles et des artefacts pour chaque activité											
Référence de la tâche	Description de la tâche	Parfaitement exécutée	Partiellement exécutée	Non exécutée	Aucune connaissance de l'existence de cette tâche	Artefacts ou Commentaires	CUS	AN	DES	PR	TL	PM	WT
SI.6.1	Affectation des tâches aux membres de l'équipe de travail selon leur rôle, en se basant sur le plan de projet actuel.		✓								✓		✓
SI.6.2	Comprendre la *Configuration logicielle.*	✓								✓	✓		
SI.6.3	Documenter la *Documentation de Maintenance* ou mettre à jour la courante.	(✓ , X)		✓						✓			
SI.6.4	Vérification de la *Documentation de Maintenance.*	(✓ , X)				O				✓	✓		
SI.6.5	Incorporation de la *Documentation de Maintenance* comme baseline pour la *Configuration logicielle.*	(✓ , X)									✓		
SI.6.6	Exécuter la livraison selon les *Instructions de livraison.*				✓								

Tableau 6.6 : Évaluation de l'activité de livraison de produit

6.2.2 Étude de compatibilité entre les processus de la norme et les processus libres

La *compatibilité* qualifie de façon générale deux éléments qui peuvent aller ensemble ou s'accorder [33]. En informatique, le terme désigne l'aptitude qu'a le logiciel à pouvoir être combiné à d'autres logiciels ou d'autres matériels. En ce qui concerne la compatibilité de deux processus, nous dirons que c'est l'aptitude qu'a un processus à pouvoir s'accorder ou à être combiné à un autre processus.

Nous avons choisi une méthode basée sur des **critères de compatibilité** aux F/OSS, car selon nous elle semble être une méthode fiable et efficace pour évaluer la compatibilité de la norme ISO/IEC 29110 selon le contexte des F/OSS. Cette approche est avantageuse car elle est conforme aux valeurs, à la culture et à la philosophie de développement des F/OSS. Compte tenu que la norme ISO/IEC 29110 comporte en tout cinq activités pour le processus d'implémentation de logiciel, selon le nombre d'activités de la norme qui seront compatibles aux F/OSS, nous pourrons donc aisément déterminer le pourcentage de compatibilité de la norme ISO/IEC 29110 à la culture, aux valeurs et même à la philosophie des F/OSS.

Afin de mener à bien la présente étude de compatibilité, nous débuterons par l'identification des différents critères relatifs à l'évaluation de la compatibilité de la norme ISO/IEC 29110 avec le mode de développement de logiciels libres. Il faut cependant garder à l'esprit la méthodologie de cette évaluation décrite au chapitre 4 – section 4.4.2.

6.2.2.1 Identification des critères de compatibilité

L'identification des différents critères de compatibilité se base sur une étude rigoureuse et approfondie des projets Linux, Apache, GNOME et Mozilla. De ce fait, quelques projets de F/OSS dans ces communautés ont retenus notre attention. Le

choix de ces sous-projets est basé sur la popularité de ces sous-projets en logiciel libre et dans leurs communautés libres respectives. Ainsi, ces sous-projets de F/OSS regorgeront d'une documentation assez dense. Par ailleurs, nous avons également basé notre choix sur la taille de ces sous-projets en termes de lignes de code et en termes du nombre de participants. Les sous-projets libres retenus sont donc: le noyau Linux, GCC, Apache HTTP Server, Firefox, GNOME. Ils sont des projets à large taille de code et possédant un nombre moyen à très grand de participants.

- **Le noyau Linux :** projet libre à large taille de code et à très grand nombre de participants. Il est le tout premier projet de logiciel libre crée en 1991 par Linus Dorval [16]. Il est le noyau du système d'exploitation libre GNU/Linux, plus connu sous le terme Linux. Consulter son site web pour des details sur les versions [21];
- **GCC :** le projet GCC est une partie du projet GNU. Il représente le compilateur du système d'exploitation GNU/Linux. C'est un projet libre à large taille de code et à très grand nombre de participants. La majorité des décisions dans ce projet sont prises par un comité de pilotage (en anglais *Steering Committee*) fondé en 1998 dans l'intention d'empêcher n'importe quelle personne ou organisation de prendre le contrôle du projet GCC [57]. Ce comité de pilotage a pour rôle de prendre la majorité des décisions dans le plus grand intérêt du projet GCC et de s'assurer que le projet GCC adhère à la majorité de ses principes fondamentaux selon la déclaration de la mission du projet [57];
- **Apache HTTP Server :** c'est un projet libre à large taille de code et à taille moyenne en nombre de participants [91];
- **Firefox :** c'est un projet de taille moyenne de code et possédant un grand nombre de participants. Sa taille en nombre de participants est quasiment semblable à celle du noyau Linux [104].
- **GNOME :** c'est un projet libre à large taille de code et à très grand nombre de participants [91].

Le projet GNOME a particulièrement suscité notre intérêt car il est lui-même un sous-projet du projet GNU/Linux. Les informations relatives aux sous-projets que nous avons choisis pour cette étude de compatibilité, ont été obtenues à partir de la revue de littérature faite au chapitre 3, de l'analyse des F/OSS faite au chapitre 5, et des informations supplémentaires trouvées sur les sites web respectifs de ces sous-projets.

En logiciels libres, nous avons remarqué une distinction notable entre les projets de F/OSS en ce qui concerne les politiques ou philosophies de pratiques d'ingénierie. Elles incluent : les processus, les rôles de participants, les outils utilisés ainsi que l'architecture. Cependant, le but pour notre étude de compatibilité est de faire ressortir les différentes similitudes ou synthèses des pratiques communes à chacun des sous-projets retenus. Ces similitudes constitueront pour la présente étude, les différents critères de compatibilité. Les questions que nous nous posons à cet effet sont essentiellement centrées sur les aspects suivants :

- **La taille des projets**. Ici, elle fait référence au nombre de personnes qui participent au projet. La question ici est : la taille des projets de F/OSS en nombre de participants au projet est elle un obstacle à l'application de la norme ISO/IEC 29110 aux F/OSS?
- **La répartition des participants au projet**. Les questions auquelles nous voulons répondre à c niveau sont les suivantes : les participants sont-ils colocalisés ou distribués géographiquement à travers le monde? Quels moyens utilisent-t-ils pour communiquer? Doivent-ils se rencontrer physiquement ou doivent-ils échanger via l'Internet? Fort heureusement, elles ont déjà été répondues dans les chapitres 3 et 5. Certaines enquêtes menées dans les F/OSS ont identifié que les participants aux F/OSS sont répartis géographiquement à travers le monde [92] et qu'ils ne se rencontrent jamais face à face, ni même par téléphone [94]. La collaboration entre développeurs se fait via les listes de diffusion (*mailing lists*) des développeurs ou à l'aide

des systèmes de partage de contenus ou des systèmes de messagerie instantanée, tels que IRC, Facebook ou encore Twitter (voir tableau 3, pour d'autres exemples). De plus, certains projets libres à l'instar d'Apache, font des réunions via Skype [97]. Mais très souvent dans les projets F/OSS, l'usage des systèmes utilisant des vidéos ou du son ne sont pas utilisés, car la plupart ne permettent pas d'archiver les communications échangées. C'est la raison pour laquelle les systèmes de messagerie, les *mailing lists* des développeurs et les systèmes de partage de contenu sont privilégiés.

- **Les pratiques de développement relatives aux projets libres retenus**. A ce niveau, les questions auxquelles nous voulons répondre sont les suivantes : comment les tâches à réaliser sont-elles réparties entre les différents participants au projet ? quelles pratiques de qualité sont utilisées dans les différents projets libres retenus? quels rôles sont associés à chaque phase du développement libre? Le chapitre 5 de cette recherche présente de façon détaillée les pratiques du développement libre.

Lors de notre observation des F/OSS en général (rappel chapitres 3 et 5), et particulièrement des sous-projets libres retenus pour cette étude de compatibilité, nous avons remarqué que chacun de ces sous-projets sont eux mêmes divisés en de plus petits de projets [21, 29, 57, 61, 95], afin de faciliter leur développement. Cela est un point pertinent car, chacun des projets au sein des sous-projets libres retenus, possède ses propres contributeurs et forme en lui-même sa propre communauté. Le nombre de participants au projet de F/OSS dans sa globalité est donc divisé selon les différents groupes de développement constituant des sous-projets au sein du projet de F/OSS global. Nous retrouvons là, la structuration d'une grande organisation de projet dont les membres sont répartis en petits groupes de développeurs par sous-projet. Sur ce point, la norme ISO/IEC 29110 et les F/OSS pourraient être conformes. De plus il est noté dans la norme qu'elle ne vise pas à empêcher ni à décourager

l'utilisation des pratiques des processus du cycle de vie qu'elle décrit, pour des organisations ou entreprises plus grandes (d'après la page 1 de la norme) .

« *The life cycle processes described in the set of ISP*[33] *and Technical Reports are not intended to preclude or discourage their usage by organizations bigger than VSEs.* » [73].

Il ne nous reste plus qu'à trouver une façon concrète de restreindre dans les F/OSS, le nombre de participants ou développeurs pour chacun des sous-projets les plus internes d'un projet parent, sans que cela ne trouble ou n'aille à l'encontre des valeurs et à la culture des F/OSS. Par exemple pour un sous-projet dans Linux, ce sous-projet est lui même subdivisé en différents modules, chacun répartis en plusieurs composants. Un composant pour un module donné de ce sous-projet, devra avoir un nombre de développeurs inférieur ou égal à 25. De cette façon, les F/OSS seront conformes au cadre conceptuel de la norme ISO/IEC 29110.

Comme exemple de répartition de projet de F/OSS en sous- projet, voyons le cas du projet Apache HTTP Server. Actuellement, Apache HTTP Server [29] est divisé selon les sous-projets suivants :

- Docs : ce sous-projet se focalise uniquement sur la documentation du projet, question de maintenir et d'améliorer la qualité de la documentation de ce projet.
- Test : ce projet est concentré sur la conception des outils de test pour le serveur Apache.
- Flood : c'est un testeur de charge axé sur le protocole http. Il peut être utilisé pour recueillir d'importants indicateurs de performance pour votre site web.

---

[33] ISP: Software Engineering International Standardized Profile [73].

- Libapreg : c'est une bibliothèque partagée avec les modules afin de manipuler les données des requêtes du client via l'API Apache.
- Modules : permet de mettre en évidence certains modules maintenus par le projet Apache HTTP Server et qui ne sont pas inclus dans la distribution de base. Parmi ces modules, l'on rencontre les modules suivants : *mod_fcgid, mod_ftp, mod_domain, mod_pop3, mod_arm4, mod_smtpd, Sandbox, mod_aspdotnet, mod_python*. Cependant, seuls les deux premiers modules, mod_fcgid et mod_ftp ont commencé à être développés.

De plus, nous avons également pu remarquer dans les différentes communautés libres (Apache, Linux, Mozilla, GNOME) sur lesquelles portent notre étude depuis le début (rappel chapitre 5) et particulièrement dans les sous-projets libres retenus dans cette étude de compatibilité [21, 29, 57, 61, 95], que ces projets libres s'efforcent d'avoir un processus de développement qui soit effectivement documenté, s'identifiant par le fait qu'il est effectivement possible de retrouver des pages d'aide en ligne relatives aux projets développés ou maintenus.

Le tableau suivant présente les pratiques importantes de qualité que nous avons observées dans les sous-projets libres retenus pour l'étude de compatibilité.

Sous-projets de F/OSS	**Pratiques de qualité notables**	
	Particularités	**Similitudes**
Apache HTTP Server	• Les changements proposés sont votés au travers de la *mailing list*. 3 votes positifs et aucun vote négatif, sont requis pour soumettre un changement. • Actuellement les bogues sont	• Les défauts majeurs sont détectés majoritairement par les utilisateurs. Les bogues sont détectés et corrigés tant par les utilisateurs que par les développeurs. • Les projets sont menés par de

	reportés via Bugzilla. • Apache a documenté chacune des versions du serveur. • Il est constitué de 6 sous-projets.	nombreux participants répartis géographiquement à travers le monde et selon leur appartenance à un module ou sous-projet au sein du projet parent. • Les bogues et les problèmes de qualité peuvent être discutés sur les *mailing lists* des développeurs (cas du noyau Linux, ce sont les *mailing lists* du noyau), ou reportés via un système de suivi de bogue, sur le portail web spécifique à chaque sous-projet (dans le cas du noyau Linux, c'est le portail web spécifique à une distribution Linux). • Le code source du projet est disponible et visible pour n'importe quel individu. • Chaque développeur et participant non-technique choisit la tâche qu'il veut réaliser. Il peut s'agir des tâches de programmation, de conception, de documentation et bien d'autres encore. • Chaque projet est divisé en de petits sous-projets afin de faciliter et d'améliorer son développement. • Chaque participant a droit au minimum aux quatre libertés.
GCC (GNU Compiler Collection)	• Le projet requiert l'usage des standards de codage GNU ainsi que certains standards de codage spécifiques à gcc. • Utilise une suite de tests très détaillée, construite autour de l'outil DejaGnu. • Il y a eu une parfaite conversion du développement des Cathédrale à celui du Bazar.	
Noyau Linux	• Plus de 50 listes de diffusion se concentrent sur le développement du noyau. • Il est constitué de 983 sous-projets. • Les bogues sont reportés via Bugzilla seulement pour la version 2.6.	
GNOME	• GNOME a documenté les feuilles de routes (*guidelines*) de programmation. • Le système est testé et construit automatiquement grâce à Autoconf.	
Firefox	• Tests de portabilité, assistés par Tinderbox. • Builds automatiques via Tinderbox.	

	• Les tests de régression sont effectués afin de compléter les tests unitaires ayant été réalisés.	

Tableau 6.7 Résumé des pratiques de quelques sous-projets libres construit de [21, 25, 29, 49, 51, 52, 57, 60, 61, 67, 83, 95, 101, 102, 104]

Au vue de la précédente étude menée dans les sous-projets libres retenus, nous remarquons que tous ces projets ont en commun certaines pratiques (voir tableau 6.7). Ces différentes similitudes représentent donc nos critères de compatibilité aux F/OSS. Nous les classons de la façon suivante :

- Les valeurs des projets libres telles qu'identifiées par Michmayr [92]:
  - **Critère 1 : les participants aux projets de logiciels libres doivent être distribués géographiquement à travers le monde** ;
  - **Critère 2 : les participants des projets de logiciels libres doivent être généralement des volontaires non payés**;

- En plus des valeurs, nous rajoutons certains aspects qui pour nous sont très importants et qui font partis des caractéristiques des logiciels libres:
  - **Critère 3 : les projets de F/OSS doivent être composés d'une multitude de personnes, habituellement, largement supérieure à 25**. En général dans les communautés libres, plus de 25 personnes peuvent participer à la réalisation d'une activité du cycle de vie de logiciel, mais sur différents fronts ou sous-projets. Les projets sur lesquels porte notre étude sont des projets à succès, possédant une très grande taille en ligne de code et dont les communautés vont d'un nombre moyen à très grand. Les projets Apache ou Mozilla sont supportés par des communautés de dizaines de centaines de développeurs et

de millions d'utilisateurs [56]. Apache est considéré dans le monde du libre comme un projet libre à large taille de code et à taille moyenne en nombre de participants [91]. Quant à Mozilla, il est un projet de taille moyenne de code et possédant un grand nombre de participants. Sa taille en nombre de participants est quasiment semblable à celle du noyau Linux [104]. Linux est considéré comme un projet libre à large taille de code et à très grand nombre de participants. Il est supporté par une communauté de milliers de millions de personnes englobant développeurs et utilisateurs. Il a été identifié que le noyau Linux possède une communauté de plus de 12 millions d'utilisateurs et qu'elle croit de 40% par année [66] (en 2003). Le noyau Linux est supporté par des milliers de développeurs [65, 66] par version [32], tout en sachant qu'il y a toujours deux types de versions, la stable et la non stable. Les différents projets ou communautés de F/OSS connus, par exemple Linux, Apache, Mozilla, GNOME et bien d'autres encore; sont divisés chacun en de petits groupes ou sous-projets. Chacun de ces sous-projets de F/OSS constituent en eux-mêmes une petite communauté, et sont eux aussi divisés en de plus petits projets afin de faciliter et d'améliorer leur développement. Cet aspect constitue un avantage de développement des F/OSS, mais il constitue aussi la cause de la difficulté d'intégration de tous ces différents modules du projet libre concerné. Par ailleurs, nous rappelons que la norme en logiciel libre, voudrait qu'il y ait en moyenne au minimum 15 développeurs de confiance pour maintenir un projet libre afin qu'il fonctionne [26] (confer chapitres 3 et 5). D'autre part, la norme ISO/IEC 29110 est prescrite pour un profil de TPOs. Cependant comme il est noté dans la norme, elle n'est pas destinée à décourager ni à empêcher son utilisation dans des organisations plus grandes que les TPOs [73]. Il convient donc, pour rendre concrètement les F/OSS conformes au profil de TPOs pour lequel la norme est prescrite, que le développement dans les F/OSS soit organisé en pools de développeurs d'au plus 25 personnes. À cet effet, pour que cela puisse être réalisable dans les

F/OSS, tout en sachant déjà qu'un projet de F/OSS est divisé en de plus petits projets, les gestionnaires de projets de F/OSS pour chaque module du projet devront s'assurer que le projet qu'ils maintiennent ne contient pas plus de 25 personnes par développement de composant. Ainsi, le nombre de participants aux projets libres ne constituera plus un problème face à la prescription de la norme ISO/IEC 29110;

- **Critère 4 : les participants doivent choisir eux-mêmes les tâches qu'ils désirent réaliser selon leur expertise ou leur niveau de confiance, conformément à la liste des tâches à réaliser.** Cette façon de procéder a été voulue telle quel en logiciel libre afin de préserver les libertés des utilisateurs;
- **Critère 5 : le code source des logiciels libres doit être disponible pour tous les utilisateurs, compte tenu des quatre libertés** [101].

Les deux premiers critères permettront de vérifier si la norme ISO/IEC 29110 est conforme aux valeurs des logiciels libres. Le troisième critère quant à lui tente de vérifier si la norme ISO/IEC 29110 peut être utilisée dans des projets à très grand nombre de participants, dans notre cas, les logiciels libres. Quant aux critères 4 et 5, ils tentent de vérifier si la norme ISO/IEC 29110 peut s'accorder au mode de développement de logiciel libre, en l'occurrence au niveau de la transparence de tout processus et document, ainsi que la disponibilité du code pour tous les membres du projet.

L'approche utilisée pour l'évaluation de la compatibilité de la norme ISO/IEC 29110 aux F/OSS (tableaux 6.8 et 6.9) et présentée en détail dans la section 6.2.2 (voir également le chapitre 4 – section 4.4.2 en rappel de la méthodologie) ne fait pas l'unanimité. Elle n'est donc pas exclusive. Cette approche se concentre sur des critères de compatibilité typiques aux F/OSS que nous avons identifiés par une observation de certains projets de F/OSS.

Les tableaux 6.8 et 6.9 représentent les différentes analyses de compatibilité de la norme ISO/IEC 29110 à la culture des F/OSS.

Critères de compatibilité	Commentaires se basant sur les prescriptions de ISO/IEC 29110
Critère 1	La norme ne mentionne pas de restriction sur la répartition géographique des membres de l'entreprise, organisation ou département qui l'utilise pour guider leur développement de logiciel. Les membres donc peuvent être centralisés ou distribués géographiquement à travers le monde comme tel est le cas dans les projets de logiciels libres.
Critère 2	La norme ne mentionne pas que les employés de l'entreprise développant des produits logiciels doivent impérativement être payés. Cela étant, il n'y a donc aucune obstruction à la norme.
Critère 3	Premièrement la norme est prescrite pour des projets constitués au maximum de 25 personnes. Et donc, compte tenu du très grand nombre de personnes participants aux projets libres, nous sommes amenés à dire que cette considération de la norme n'est pas respectée. Par ailleurs, il est mentionné dans la norme qu'elle n'est pas destinée à décourager ni à empêcher l'utilisation des pratiques des processus du cycle de vie décrit, pour des organisations ou entreprises plus grandes (voir page 1 de la norme). De ce fait, nous avons proposé une répartition du développement des F/OSS en pools de développeurs dont le nombre est inférieur ou égal à 25. Pour conclure, il n'y a donc aucune obstruction à la norme.
Critère 4	La norme suggère pour le cycle de vie de logiciel, que le « lead » technique assigne les tâches aux membres de l'équipe selon leur rôle. En logiciel libre, chaque développeur choisit la tâche qu'il désire réaliser en rapport à son expertise. Ce critère n'est dont pas respecté par la norme.
Critère 5	Aucune mention n'est faite dans la norme quant à la transparence ou disponibilité du code source pour tous les participants au projet. Cela étant, il n'y a donc aucune obstruction à la norme.

Tableau 6.8 Analyse générale de l'applicabilité de la norme ISO/IEC 29110 avec le mode de développement de logiciels libres

Critères de compatibilité	Commentaires se basant sur les pratiques de activités de ISO/IEC 29110					
	Initiation d'implémentation	**Analyse des exigences**	**Architecture et conception**	**Construction logicielle**	**Intégration et tests**	**Livraison de produit**
Critère 1	La norme ne mentionne pas de restriction sur la répartition géographique des membres de l'entreprise, organisation ou département qui l'utilise pour guider leur développement de logiciel. Les membres peuvent donc être centralisés ou distribués géographiquement à travers le monde comme tel est le cas dans les projets de logiciels libres. Donc, il n'y a aucun compromis de la norme avec la répartition des participants dans les projets de F/OSS.					
Critère 2	La norme ne mentionne pas que les employés de l'entreprise développant des produits logiciels doivent impérativement être payés. Cela étant, il n'y a donc aucune obstruction à la norme. Aucun compromis de la norme avec le bénévolat des participants aux projets de F/OSS.					
Critère 3	Dans sa prescription dans la norme, l'activité comprend 3 rôles faisant référence à au plus 25 personnes. Habituellement en logiciel libre, l'initiation d'implémentation	Dans sa prescription dans la norme, l'activité comprend 4 rôles faisant référence à au plus 25 personnes. Dans les F/OSS, les exigences sont définies par les programmeurs et prennent forme à travers de multiples	Dans sa prescription dans la norme, l'activité comprend 3 rôles faisant référence à au plus 25 personnes. En logiciel libre, ce nombre dépasse grandement 25. Donc la norme est non conforme aux F/OSS.	Dans sa prescription dans la norme, l'activité comprend 2 rôles faisant références à au plus 25 personnes. Cependant en F/OSS le nombre de personnes participant à cette activité par module, est grandement supérieur	Dans sa prescription dans la norme, l'activité comprend 4 rôles faisant référence à au plus 25 personnes. Dans les F/OSS, l'intégration du logiciel est habituellement réalisée par le groupe de développeurs de confiance ou mainteneurs de projets.	Dans sa prescription dans la norme, l'activité comprend 3 rôles faisant référence à au plus 25 personnes. Cependant, en logiciel libre, se sont uniquement les développeurs de confiance qui effectuent cette activité. Hors leur nombre est en moyenne au moins égal à 15 [26]

	de logiciel provient de l'un des membres des développeurs de confiance du projet. Leur nombre est en moyenne au moins égal à 15 [26] et peut dépasser 25 personnes. Donc, la norme est non conforme aux F/OSS. Pour une conformité parfaite à la norme et une meilleure gestion ou une efficacité dans la coordination des	discussions entre développeurs via la liste de diffusion. Ces développeurs jouent plusieurs rôles au sein du projet libre, en rapport à leur mérite dans la communauté. Ils dépassent généralement 25. La norme n'est donc pas conforme aux F/OSS. Aucune modification du nombre de participants à ce niveau n'est envisageable dans le contexte des F/OSS.		à 25. Mais ces participants sont répartis selon les différents sous-projets qui composent le projet parent. Pour chacun de ces sous-projets, le nombre de participants est habituellement supérieur à 25. En somme, la prescription de la norme est non conforme aux F/OSS.	En moyenne ce groupe est constitué d'au moins 15 personnes [26] et peut dépasser 25 personnes. Concernant l'activité de tests dans les F/OSS, n'importe quel développeur peut y participer. Ces participants ou développeurs sont répartis selon les différents sous-projets qui composent le projet parent. À la fin de l'activité de tests, ce sont les mainteneurs qui décident de la solution à retenir. En somme, la prescription de la norme est non conforme aux F/OSS.	et peut dépasser 25 personnes. Donc, la norme est non conforme aux F/OSS. Pour une conformité parfaite à la norme et une meilleure gestion ou une efficacité dans la coordination des efforts à ce niveau dans les F/OSS, ce nombre peut être fixé au maximum à 25. Ce qui garantira d'une part les chances que ce projet réussisse et d'autre part qu'il soit conforme à la norme.

	efforts à ce niveau dans les F/OSS, ce nombre peut être fixé au maximum à 25. Ce qui garantira d'une part les chances que ce projet réussisse et d'autre part qu'il soit conforme à la norme.	Le processus des exigences tel que réalisé dans les F/OSS est une source de la qualité élevée dans les F/OSS. Il permet de détecter et de corriger des défauts bien avant la phase de tests, lesquels pourraient être dus à des exigences mal définies.			Pour une conformité parfaite à la norme et une meilleure gestion ou une efficacité dans la coordination des efforts à ce niveau dans les F/OSS, ce nombre peut être fixé au maximum à 25. Ce qui garantira d'une part les chances que ce projet réussisse et d'autre part qu'il soit conforme à la norme.	
Critère 4	Assignation des tâches aux membres de l'équipe non prescrite à ce niveau dans la norme. Prescription non	Prescription de la norme, non conforme aux logiciels libres.	Prescription de la norme, non conforme aux logiciels libres.	Prescription de la norme, non conforme aux logiciels libres.	Prescription de la norme, non conforme aux logiciels libres.	Prescription de la norme, non conforme aux logiciels libres.

	conforme aux logiciels libres.					
Critère 5	L'usage du code n'est pas nécessaire à ce niveau dans la norme. Donc, nous ne saurons nous prononcer.	L'usage du code n'est pas nécessaire à ce niveau dans la norme. Donc, nous ne saurons nous prononcer.	Pas de compromis des prescriptions de la norme quant à la disponibilité du code dans les projets de F/OSS.	Pas de compromis des prescriptions de la norme quant à la disponibilité du code dans les projets de F/OSS.	Pas de compromis des prescriptions de la norme quant à la disponibilité du code dans les projets de F/OSS.	Pas de compromis des prescriptions de la norme quant à la disponibilité du code dans les projets de F/OSS.

Tableau 6.9 Matrice de comparaison des activités de ISO/IEC 29110 conformément aux critères de compatibilité avec les logiciels libres

6.3 PRÉSENTATION DES RÉSULTATS

6.3.1 Écarts observés entre les processus libres et processus classiques de développement

Dans cette section les écarts observés en pratique lors du processus de développement de logiciels libres sont présentés, conformément aux activités de la norme ISO/IEC 29110.

➢ **Présentation des écarts**

Sigles et titres de l'activité ISO/IEC 29110		Écarts observés du processus de développement de logiciels libres au niveau :			
		Activités	**Tâches spécifiques**	**Produits intermédiaires**	**Rôles des individus**
SI.01	**Initiation et implémentation du logiciel**	Cette activité est réalisée partiellement.	Écarts observés dans : -La compréhension commune du plan de projet et la prise des engagements: il n'existe pas de plan de projet, cependant il est implicite à chaque projet et prend forme durant l'implémentation du logiciel. - Pas de contrainte de coût de projet. -Pas de composition de l'équipe de travail, car l'adhésion au projet se fait de manière volontaire. De plus, les contributeurs ayant fournis plus de contributions	Il n'existe pas de plan de projet.	Il n'existe pas vraiment d'écarts observés. La seule différence réside du fait que le LT dans le monde du libre porte également le chapeau de PM.

			significatives sont ceux sur qui la communauté compte en premier.		
SI.02	**Analyse des exigences du logiciel**	C'est une activité très structurée, cependant, faite partiellement conformément à la norme. L'observation de cette activité montre que la grande majorité des pratiques de la norme ISO/IEC 29110, est effectuée dans cette activité, ceci à quelques détails près.	Des écarts existent au niveau des tâches suivantes : -Chaque développeur choisit la tâche sur laquelle il désire travailler selon son expertise, ceci suivant la planification courante des tâches à réaliser, présente sur le site web de sa communauté. -Les exigences apparaissent lors d'un débat par courriel dont l'intention originelle n'était pas de faire une analyse des besoins; jusqu'à leur stabilisation à peu près dans une centaine de conversations. -Le client ne participe généralement pas à la	-Non existence de plan de projet. -Dans bon nombre des cas, absence de l'artefact documentation utilisateur du logiciel. Cependant, en F/OSS, pour tous les projets retenus, il est possible de retrouver des pages d'aide en ligne, servant de documentation utilisatrice du logiciel. -Nous n'avons pas trouvé des mentions sur les documents de résultats de vérification des exigences et de résultats de validation des exigences. Cependant, ils peuvent être retracés à partir	Les rôles observés sont : -TL -WT : chaque programmeur dans l'équipe de travail, joue également le rôle d'analyse. <u>Écarts :</u> -Le programmeur définit les exigences et la *documentation utilisatrice de logiciel* (tâche qui n'est cependant pas obligatoire); -les leaders techniques, qui sont eux-mêmes des programmeurs, vérifient

			validation des exigences du logiciel. Cependant dans certains cas, ils peuvent influencer les décisions concernant les exigences [60].	des discussions faites sur les exigences logicielles présentent sur la liste de diffusion	les exigences du logiciel, jusqu'à ce qu'elles soient consistantes.
SI.03	**Architecture et conception**	Particulièrement dans Mozilla, cette activité est exécutée partiellement et les pratiques observées dans Mozilla ressemblent beaucoup à celles de la norme ISO/IEC 29110.	Les écarts observés sont les suivants : -Tout développeur désirant participer à la réalisation de cette activité, s'assigne lui-même une tâche. -Dans Mozilla, il n'est nulle part mentionné que la conception logicielle, cas et procédures de tests, ainsi que document de traçabilité sont intégrés dans la configuration logicielle. Cependant, nous savons que les projets de F/OSS utilisent le CVS, donc ces documents pourraient être retrouvés par filature ou suivi	Dans la majorité des cas, non existence de document de conception de logiciel. Il y a cependant une contradiction faite par Reis et Fortes [31] car bon nombre d'artefacts présents dans la norme, ici « *le document de conception logicielle, le document de requête de changements, le document de tracabilité* », sont présents dans le processus de développement de Mozilla.	Pas d'écart réel observé (cadre de Mozilla). Ici, les participants de cette activité revêtent plusieurs chapeaux

			des discutions et par suivis des bogues. Dans le cas échéant, ils pourraient être retrouvées dans les archives de documents du projet, contenues dans le CVS.	De façon générale dans les F/OSS, il y a une traçabilité moyenne du suivi des problèmes de conception [98]. Ainsi, les écarts suivants sont observés : -Inexistence du document de procédures de tests. Cependant peut être construit à partir des archives du projet contenues dans le CVS - Pas de document de configuration logicielle	
SI.04	**Construction du logiciel**	L'activité de construction de logiciel dans le monde du libre n'est pas totalement conforme à la norme. Elle est exécutée partiellement,	Les écarts sont les suivants : -Chaque programmeur choisit la tâche sur laquelle il désire travailler, selon son expertise.	Aucun écart observé. De plus, la majorité des F/OSS gardent trace des défauts de code [98].	Aucun écart observé

		cependant elle est très structurée et les tâches réalisées sont effectuées avec beaucoup de finesse.			
SI.05	**Intégration et tests**	Cette activité est partiellement réalisée.	-Tout développeur désirant participer à la réalisation de cette activité, s'assigne lui-même une tâche. -La mise à jour du rapport de traçabilité n'est pas une tâche obligatoire.	-Il n'existe pas de document formel de traçabilité ni de rapport de test, cependant, ils peuvent être construits à partir des informations se trouvant dans la liste de diffusion des développeurs et dans le CVS. - Le guide d'opérations du produit logiciel n'est presque pas existant en logiciel libre.	-Le leader technique intègre le logiciel et définit les cas de tests pour l'intégration.
SI.06	**Livraison de produits**	Cette activité est réalisée non conformément aux pratiques prescrites en génie logiciel.	- Tout développeur désirant participer à la réalisation de cette activité, s'assigne lui-même une tâche. -La tâche de documentation ou de mise à jour de la documentation de maintenance, ainsi que la	-Habituellement, pas de Documentation de maintenance.	-Le programmeur met à jour la Documentation de maintenance.

			tâche de vérification de la documentation de maintenance, sont presque inexistantes, faute de la liberté des participants à choisir les tâches sur lesquelles ils désirent travailler.		

Tableau 6.10 Écarts observés du développement de logiciels libres par rapport au développement classique de logiciels

- **Différences entre les processus libres et processus d'ingénierie : Forces, faiblesses et pratiques de qualité**

Modes de développement	Différences entre les processus		
	Forces	**Faiblesses**	**Pratiques de gestion de qualité**
Mode de développement de F/OSS	Présence d'une infrastructure solide, soutenue par des outils de : contrôle et gestion de versions, de construction du logiciel, de suivis de bogues, etc. Code source et documents disponibles pour tous les participants au projet.	Non-conformité des normes de développement. À la place, usage des styles de codages propres à chaque projet. Pas de plan de projet.	Travail choisi par les membres de l'équipe de travail. Les exigences sont définies par les programmeurs. Un développement pas souvent

	Usage de la modularité du code et de la réutilisation du code des composants. Coût de développement faible car les participants sont majoritairement des bénévoles non payés.		documenté ou défini. Méthodologie de développement proche des méthodes agiles. Détection et correction rapide des défauts, bien avant la phase d'intégration et des tests. Maintenance et évolution continues du logiciel par les programmeurs. Revues récurrentes par les pairs. Le logiciel est intégré par les leaders techniques (ce sont les mainteneurs de projets). Possibilité de reconstruire le processus de prise de décisions car il est archivé en ligne [91].
Mode de développement classique de logiciels	Usage des normes de développement. Présence de plan de projet.	Quelques fois, existence de débordement de coût et des délais.	Le travail est assigné aux membres de l'équipe de travail.

	Usage d'outils facilitant le développement.		Les exigences sont définies par l'analyste. Une méthodologie de développement bien définie et des processus structurés. Maintenance et évolution sous la demande des clients. Le logiciel est intégré par les programmeurs.

Tableau 6.11 Forces, faiblesses et pratiques de qualité entre le développement libre et le développement en ingénierie

6.3.2 Résultats de l'étude de compatibilité

Nous remarquons selon la précédente analyse (tableau 6.8) que les critères 1, 2, 3 et 5 sont respectés, ceci au détriment du critère 4. Cette analyse permet de savoir si la norme ISO/IEC 29110 peut s'appliquer aux F/OSS. Nous dirons que la norme ISO/IEC 29110 satisfait à la majorité des critères de compatibilité aux F/OSS à l'exception du critère 4. Elle serait donc compatible aux F/OSS à 80%.

Cependant, l'analyse de compatibilité faite au tableau 6.8 n'est pas suffisante pour identifier les activités de la norme ISO/IEC 29110 qui sont compatibles et celles incompatibles aux F/OSS. Ainsi, une analyse supplémentaire a été faite (voir tableau 6.9), laquelle présente l'évaluation des différentes activités de la norme ISO/IEC 29110 par rapport aux critères de compatibilité aux F/OSS. Les résultats de l'étude de compatibilité de la norme ISO/IEC 29110 avec le mode de développement de logiciel libre sont représentés dans le tableau 6.12, précisément concernant l'identification des activités compatibles et celles incompatibles aux F/OSS.

Le signe "+" signifie que l'activité est compatible à un critère et le signe "-" signifie que l'activité est non compatible à un critère. Le signe "*" signifie que le critère de compatibilité aux logiciels libres est non défini dans la norme. Toute activité de la norme contenant le signe "*" sera d'office exclue car, nous ne pourrons pas l'évaluer totalement par rapport aux 5 différents critères de compatibilité aux F/OSS. Étant donné que nous avons identifié 5 critères de compatibilité aux F/OSS pour évaluer la norme ISO/IEC 29110, intuitivement, une activité de la norme ISO/IEC 29110 sera dite compatible aux F/OSS si elle satisfait à la majorité des critères de compatibilité. À cet effet, un minimum de trois "+" et aucun "*" lors de l'évaluation d'une activité de la norme ISO/IEC 29110, est donc suffisant pour nous permettre de déterminer que cette activité est compatible aux F/OSS.

Critères / Processus de ISO/IEC 29110	1	2	3	4	5	Résultats
Initiation d'implémentation	+	+	+	*	*	Non compatible
Analyse des exigences	+	+	-	-	*	Non compatible
Architecture et conception	+	+	-	-	+	Compatible
Construction logicielle	+	+	-	-	+	Compatible
Intégration et tests	+	+	+	-	+	Compatible
Livraison de produit	+	+	+	-	+	Compatible

Tableau 6.12 Résultats de l'étude de compatibilité de la norme ISO/IEC 29110 conformément aux logiciels libres

En somme, il existe une majorité de processus de la norme qui sont compatibles avec le mode de développement de logiciel libre, particulièrement les processus : d'architecture et conception, de construction logicielle, d'intégration et tests, et enfin de livraison du produit. Étant donné que 4 sur 6 des processus d'implémentation de la norme sont compatibles aux F/OSS selon leurs valeurs et leur culture, **le pourcentage réel** de la compatibilité de la norme ISO/IEC 29110 aux F/OSS est donc de **66, 66 %.**

6.4 RÉSUMÉ DU CHAPITRE

Nous avons commencé par faire une brève présentation de la norme ISO/IEC 29110. Par la suite, nous avons procédé à deux analyses de processus de développement de logiciels. La première étant l'analyse ou l'évaluation du processus de développement de logiciels libres et la seconde portant sur l'analyse des processus prescrits par la norme ISO/IEC 29110. La première analyse nous a permis de déceler de nombreux points d'écarts entre le développement libre et le développement en génie logiciel, de même que les forces et les faiblesses de chacun de ces processus de développement. Ainsi, nous avons pu remarquer que la majorité de ces points d'écarts entre le processus de développement libre et le développement classique de logiciels, se situe au niveau des activités d'analyse des exigences, d'architecture et de conception, et l'activité d'intégration et de tests. Ces écarts sont principalement des écarts de documentation. Ils sont principalement causés d'après notre étude des F/OSS, par le bénévolat des participants aux projets de F/OSS. Par ailleurs, une autre raison pourrait être le manque d'une étape de vérification et de validation.

La seconde analyse a fait appel à des critères de compatibilité aux logiciels libres, que nous avons soigneusement identifiés à travers l'étude des pratiques de qualité utilisées dans les sous-projets libres suivants : Apache HTTP Server, le noyau Linux, GCC, Firefox et GNOME. Ces critères de compatibilité aux F/OSS ont été la base de notre étude de la compatibilité de la norme ISO/IEC 29110 aux F/OSS.

Ce chapitre nous a ainsi mené à relever les différences entre le développement de logiciels libres et le développement en génie logiciel, de même qu'à l'identification des activités ou processus de la norme ISO/IEC 29110 qui sont compatibles et celles qui sont non compatibles aux logiciels libres. Nous noterons tout de même que seuls les activités de la norme ISO/IEC 29110 qui sont compatibles avec les logiciels libres, nous intéresseront pour la suite de notre étude.

CHAPITRE VII

ANALYSE ET RÉFLEXIONS SUR LES RÉSULTATS

7.1 INTRODUCTION

Les comparaisons faites dans le chapitre précédent, nous ont mené à deux principaux résultats. Le premier résultat présente les différences entre le développement libre et le développement classique de logiciels. Le second résultat nous a permis d'identifier les activités de la norme ISO/IEC 29110 qui sont compatibles aux F/OSS.

Dans ce chapitre, nous discuterons de la compatibilité de la norme ISO/IEC 29110 avec les logiciels libres. Cet aspect de compatibilité aux F/OSS est très intéressant, du fait qu'il nous permettra d'introduire la façon dont la norme ISO/IEC 29110 pourrait être appliquée aux logiciels libres afin d'améliorer leur processus de développement, compte tenu des différentes valeurs encourues dans les F/OSS. À cet effet, nous procéderons à une analyse et à une réflexion sur chaque résultat de compatibilité obtenu.

7.1.1 Rappel des résultats obtenus : synthèse des résultats

7.1.1.1 Différences entre le développement de logiciels libres et le développement classique de logiciels

La synthèse des différences relevées entre le processus de développement de logiciels libres et le développement de logiciels en ingénierie, est représentée par le tableau 7.1.

	Développement fermé de logiciels	Développement de F/OSS
Différences globales	Le travail est assigné aux membres de l'équipe de travail.	Travail choisi par les membres de l'équipe de travail.
	Les exigences sont définies par l'analyste.	Les exigences sont définies par les programmeurs.
	La méthodologie de développement est bien définie et les processus sont structurés.	Un développement peu documenté ou pas souvent défini. La méthodologie de développement se rapproche des méthodes agiles.
	Les défauts sont majoritairement détectés et corrigés au niveau de la phase de tests et d'intégration.	Une détection et correction rapide des défauts, bien avant la phase d'intégration et des tests.
	Maintenance et évolution sous la demande des clients.	Maintenance et évolution continue du logiciel par les programmeurs. Revues récurrentes par les pairs.
	Le logiciel est intégré par les programmeurs.	Le logiciel est intégré par les leaders techniques.
	Composition de l'équipe de travail qui participera au développement d'un projet. Et le travail est assigné à chacun des membres par le	Chaque personne intéressée par le projet en cours de développement adhère par sa propre volonté, et choisit les tâches qu'elle désire réaliser en rapport à

		leader technique.	son expertise.
Particularités au niveau de chaque activité du cycle de vie du logiciel	**Initiation d'implémentation**	Existence du plan de projet	Majoritairement, l'état du travail est difficile à définir. D'où la non existence du plan de projet. Il prend forme tout au long de l'implémentation du logiciel.
	Analyse des exigences	Les exigences sont définies par l'analyste. La documentation utilisatrice du logiciel est définie par l'analyste. Le client participe à la validation des exigences du logiciel. Les exigences proviennent des besoins des clients.	Les exigences sont définies par les programmeurs. La documentation utilisatrice du logiciel si elle existe, est définie par le programmeur et est sous forme de page d'aide en ligne. Les clients ne participent pas à la validation des exigences, cependant, ils peuvent influencer les décisions prises à ce niveau. Les exigences proviennent du besoin d'un développeur à implémenter une fonctionnalité. Elles prennent forme à travers des discussions ad hoc entre

			les développeurs. Elles sont implicites à l'implémentation et à la conception du logiciel.
		Existence des documents de résultats de vérification et de validation des exigences du logiciel.	Les documents de résultats de vérification et de validation des exigences du logiciel sont inexistants dans les F/OSS. Cependant, ils peuvent être construits à partir des discussions dans les listes de diffusion des développeurs concernant les exigences du logiciel.
	Architecture et conception	Processus d'architecture et de conception de logiciels bien défini et très structuré.	Non existence d'une phase de conception logicielle explicite. Elle est implicite au développement. De même, l'architecture du logiciel prend forme au cours du cycle de vie du logiciel. Habituellement la majorité des F/OSS, procède à une traçabilité moyenne de suivi des problèmes dans la conception.
		L'analyste et le concepteur définissent ou mettent à jour la conception logicielle.	Dans le cas échéant, le programmeur définit ou documente la conception du logiciel.
	Construction logicielle	Processus bien documenté.	Pas de réelles différences. Néanmoins, en ce qui concerne la documentation du processus, ce n'est pas une tâche obligatoire pour les participants au projet. Si une telle documentation existe, elle est faite par les

			programmeurs. Tous les artefacts prescrits en ingénierie peuvent être retrouvés à partir des documents et informations se trouvant dans le dépôt de document via le CVS.
	Intégration et tests	Le logiciel est intégré par les programmeurs. Processus structuré et bien documenté	Le leader technique, connu sous le terme de mainteneur de projet dans les logiciels libres, intègre le logiciel et définit les cas de tests. Problèmes de documentation car ce n'est pas une tâche obligatoire. Ainsi, non existence de document formel de traçabilité et du rapport des tests.
	Livraison de produit	La documentation de maintenance est documentée et vérifiée par le designer.	La documentation de maintenance n'est pas souvent définie. Si elle existe, elle est mise à jour et vérifiée par le développeur et elle est disponible sous forme électronique.

Tableau 7.1 Synthèse des différences entre les logiciels libres vs développement d'ingénierie

7.1.1.2 Compatibilité de ISO/IEC 29110 avec les logiciels libres

Les résultats de compatibilité de la norme ISO/IEC 29110 avec les logiciels libres présentés au chapitre précédent, sont représentés sous forme de synthèse par le tableau 7.2. En rappel, nous avons considéré qu'une activité de la norme ISO/IEC 29110 est compatible aux F/OSS si elle satisfait à la majorité de nos critères de compatibilité identifiés à la section 6.2.2.1 du chapitre 6. Pour les détails des résultats de compatibilité de la norme ISO/IEC 29110 aux F/OSS, voir chapitre 6 – section 6.3.2.

Processus de ISO/IEC 29110	Résultats de compatibilité aux F/OSS
Initiation d'implémentation de logiciels	Non
Analyse des exigences	Non
Architecture et conception	Oui
Construction logicielle	Oui
Intégration et tests	Oui
Livraison de produit	Oui

Tableau 7.2 Synthèse des résultats de compatibilité de la norme ISO/IEC 29110 aux logiciels libres

7.2 ANALYSE DES RÉSULTATS

Bien que l'industrie du logiciel a eu à reconnaître l'intérêt des F/OSS de par la qualité des produits logiciels développés ou maintenus, ainsi que de par leur avantage économique face au développement de logiciels commerciaux, elle reste cependant

encore un peu sceptique quant au suivi des normes d'ingénierie dans le processus de développement libre. Les F/OSS sont connus par leur nature non-organisée et non-structurée, et sont majoritairement constitués de participants bénévoles. Par conséquent, la façon de procéder des F/OSS et la nature distribuée de leur développement, rendent difficile l'application des normes dans leur développement. Ce constat préoccupe grandement l'industrie [91], d'autant plus qu'elle dépend grandement des logiciels produits par les communautés de F/OSS. Par ailleurs, le modèle de développement des F/OSS est soutenu par une infrastructure solide pour un développement de logiciel de manière collaborative. Cela entraine la considération de nouvelles exigences dans les F/OSS, notamment, la fiabilité et la durabilité [120]. Ces nouvelles exigences selon Michlmayr [91], pourraient orienter les F/OSS vers une méthodologie de développement plus organisée. Ainsi certaines entreprises commerciales afin de s'assurer de la fiabilité et de la durabilité des produits logiciels libres dont elles ont besoin, tout en se conformant à la culture des F/OSS (par exemple Netscape ou encore les compagnies IBM, Sun Microsystems, Red Hat [87]), consacrent une partie de leurs ressources à la participation active au développement de ces produits logiciels dans les F/OSS (cas Apache, Mozilla, GNOME).

Notre analyse du premier résultat obtenu lors du chapitre 6 nous a permis de noter de nombreux points d'écarts entre les processus[34] libres et les processus classiques de développement de logiciels. Nous avons ainsi pu remarquer que les processus libres les plus problématiques, en d'autres termes, les processus libres possédant un plus grand nombre d'écarts par rapport à ceux du génie logiciel, sont les processus d'analyse des exigences, d'architecture et de conception, et le processus d'intégration

---

[34] Dans le contexte libre, une activité de développement est perçue comme un processus ou phase dans le processus de développement libre. Par contre dans le cadre de la norme ISO/IEC 29110, il y a deux principaux processus : le processus de gestion de projet et le processus d'implémentation de logiciel. Chacun de ces processus est constitué de multiples activités. Dans le cadre de ce travaille de recherche, nous employons souvent le terme «**processus**» dans le processus de développement libre pour désigner une «**activité**» de développement.

et de tests. De notre analyse de ce résultat, nous avons identifié que ces écarts sont principalement des écarts de documentation. D'après notre étude des F/OSS, ces écarts sont principalement causés par le bénévolat des participants aux projets de F/OSS. Par ailleurs, une autre raison pourrait être le manque d'une étape de vérification et de validation pour chacun des documents des processus libres.

L'utilisation de normes d'ingénierie tout au long d'un cycle de vie de logiciel par de nombreuses organisations, permet d'améliorer la productivité du logiciel, de réduire le coût de développement, ainsi que d'augmenter la qualité du produit logiciel développé ou maintenu. Les F/OSS gagneraient donc énormément à appliquer la norme ISO/IEC 29110 dans leur processus de développement. Ainsi par le suivi des processus prescrits par la norme ISO/IEC 29110, nous nous attendons à ce que le cycle de développement des F/OSS soit amélioré par un ajout considérable de documentation, qui selon nous, leur permettrait d'élargir leurs communautés et d'attirer de plus en plus d'utilisateurs et d'entreprises commerciales. D'autre part, l'usage de la norme ISO/IEC 29110 dans les processus libres via l'approche proposée, va permettre aux F/OSS d'avoir un développement un peu plus organisé, qui contribuera à ce que les F/OSS satisfassent les nouvelles exigences de *fiabilité* et de *durabilité* de leur mode de développement identifiées par Weber [120]. Ce qui favorisera grandement les industries logicielles à se fier aux produits logiciels libres et à les utiliser de plus en plus. De plus, cette application permettra à n'importe quelle communauté de F/OSS de garantir, de maintenir ou d'améliorer ce niveau de qualité élevée, déjà présent dans certains produits de F/OSS.

Par la suite, dans notre analyse des résultats, en occurrence, l'analyse des résultats de compatibilité de la norme ISO/IEC 29110 avec les logiciels libres, nous ne nous attarderons que sur les processus du cycle de vie de logiciels prescrits par la norme qui sont compatibles aux F/OSS. L'analyse à ce niveau consistera à identifier les

avantages de chacun de ces processus par rapport à la culture et à la philosophie de développement des F/OSS.

7.2.1 Architecture et conception

Les résultats obtenus relatant les différences entre le développement libre et celui en ingénierie nous ont permis de remarquer que cette activité est l'une des activités du processus de développement libre possédant un très grand nombre de points d'écarts avec le développement en génie logiciel, en occurrence telle que la norme ISO/IEC 29110 le suggère. L'architecture et la conception au sens du génie logiciel permettent de modéliser les exigences de logicielles afin de produire ultérieurement le code. De ce fait, réaliser une bonne architecture et une bonne conception du logiciel, permettrait aux développeurs de produire un code de qualité élevée. Par ailleurs, nous savons déjà que certains projets de F/OSS développent du logiciel de qualité. Ce qui nous pousse à nous interroger sur la nécessité du suivi des normes de base de développement sur ce processus libre. Nous analyserons les contours de cette question dans le paragraphe suivant.

Les bénéfices de cette activité selon la norme ISO/IEC 29110 pour les F/OSS sont les suivants : *Un processus bien documenté se traduisant par des artefacts définis, documentés ou mis à jour.* Les F/OSS pourraient bénéficier énormément de structurer et de définir de manière explicite l'activité de conception de logiciel. Cependant compte tenu des valeurs des F/OSS, les projets sont majoritairement développés, maintenus et gérés par des bénévoles répartis géographiquement à travers le monde. Par conséquent, structurer cette activité sera très contraignant pour les participants aux F/OSS, car elle va à l'encontre de la liberté voulue dans le déroulement des processus libres. De plus, nous savons que le développement des F/OSS est incrémental [96], que les exigences du logiciel dans les F/OSS sont définies par les programmeurs [26] et qu'elles émanent généralement des discussions entre

développeurs [107] et non des besoins d'un client. Il devient de ce fait difficile de procéder à une conception globale du logiciel, encore moins à une conception détaillée du logiciel, ni même de définir une architecture du système, sans que toutes les exigences n'aient été parfaitement définies et précises.

7.2.2 Construction de logiciel

L'activité de construction de logiciel est une activité importante. C'est au cours de cette activité que le logiciel prend forme. Les résultats obtenus relatant les différences entre le développement libre et celui en ingénierie nous ont mené à la conclusion que cette activité est conforme à celle en génie logiciel, en occurrence tel que suggérée par la norme ISO/IEC 29110. Nous avons cependant relevé quelques différences (voir chapitre 6 – section 6.3.1) qui sont les suivantes:

- Chaque développeur choisit la tâche à réaliser en rapport à son expertise;
- La documentation de ce processus n'est pas une tâche obligatoire. Cependant dans la plupart des projets libres étudiés, elle est présente : le rapport de traçabilité (pour les corrections des défauts du code).

Nous savons déjà que cette activité telle que prescrite par la norme ISO/IEC 29110 est compatible aux F/OSS. Le suivi de cette activité selon l'ISO/IEC 29110 dans le processus de F/OSS, en rapport avec leurs pratiques et leurs valeurs, devrait permettre **d'améliorer la documentation du code**. En ce qui concerne la documentation sur la configuration du logiciel, elle est déjà présente en ligne pour les projets libres étudiés.

7.2.3 Intégration et tests

Les F/OSS ont un énorme avantage par rapport au logiciels propriétaires, dû au fait que les défauts du logiciel soient détectés et corrigés par de nombreuses personnes (combinaison de développeurs et d'utilisateurs) bien avant la phase d'intégration du

logiciel. Cette façon de procéder permet d'accroitre la détection et la correction des défauts, mineurs comme majeurs du produit logiciel.

En plus de cet avantage des F/OSS sur les logiciels propriétaires, si les F/OSS utilisent la norme ISO/IEC 29110 dans leur processus de développement, ils bénéficieront de :

- La compréhension, la définition et l'exécution des cas et procédures de tests pour l'intégration du logiciel;
- L'amélioration de la qualité de chaque composant du logiciel avant de l'intégrer au logiciel;
- L'amélioration de la documentation du processus. Les artefacts sujets à cette documentation sont : le plan de test, le rapport de test et la configuration logicielle.

7.2.4 Livraison de produit

La livraison du produit est une tâche importante du processus de développement de tout logiciel. Elle consiste à fournir un logiciel intégré ainsi que toute la documentation ou les documents nécessaires à la bonne marche, à la compréhension et à l'utilisation du logiciel; aux utilisateurs. La livraison du produit dans les F/OSS selon notre observation, consiste à fournir une version emballée du logiciel parfaitement intégré et fonctionnel (connu sous le terme anglais Package), axée sur les critères d'accessibilité et de visibilité [49], question d'attirer de nombreux utilisateurs tant techniques que non-techniques [60, 83]. La documentation de maintenance de ce produit logiciel ne fait généralement pas partie du package du logiciel. Cependant, si elle existe, elle est habituellement présente sous forme de page d'aide en ligne.

L'usage de la norme ISO/IEC 29110 permettra d'améliorer la livraison de produit logiciel libre notamment par :

- La définition ou la mise à jour de la documentation de maintenance du logiciel;
- L'amélioration du document de configuration de logiciel. Si ce document n'a pas été défini, la norme suggère de le définir.

7.3 RÉFLEXION SUR LES RÉSULTATS

Avant d'aller plus loin, rappelons les objectifs que nous nous sommes fixés pour ce mémoire au chapitre 2. Ils sont les suivants :

- ✓ Identifier les problèmes de qualité que rencontrent certains projets de F/OSS;
- ✓ Analyser le processus de développement de logiciels libres :
  - o Déterminer en quoi le développement des logiciels libres se distingue du développement classique, ici, celui en génie logiciel (respect des normes de l'industrie);
  - o D'identifier lesquelles des activités de la norme ISO/IEC 29110 sont incompatibles avec la philosophie de F/OSS et les considérations autour des F/OSS, et lesquelles pourraient améliorer le processus de développement de F/OSS;
- ✓ Présenter la synthèse des résultats obtenus;
- ✓ Présenter les conclusions tirées de cette étude.

À ce stade de notre étude, nous avons atteints les trois premiers objectifs de recherche que nous nous sommes fixés. Au chapitre 3, nous avons identifié les problèmes de qualité que rencontrent les F/OSS. Au chapitre 6, nous avons évalué le processus de développement de logiciels libres et avons identifié les différences entre le processus de développement libre et le développement classique de logiciels. Par la suite, nous

avons identifié dans le même chapitre, les activités de la norme ISO/IEC 29110 qui sont compatibles et celles non compatibles aux F/OSS. Quant au dernier objectif de ce travail de recherche, il ne pourra être atteint qu'à la fin du mémoire.

L'analyse des différences entre le développement libre et le développement classique de logiciels, nous a permis d'identifier les activités du processus de développement libre possédant le plus grand nombre d'écarts avec celles du développement classique de logiciels. Ces activités sont les *activités d'analyse des exigences, d'architecture et de conception, et l'activité d'intégration et de tests*. De plus, l'étude de compatibilité faite au chapitre 6, a permis d'identifier les activités de la norme ISO/IEC 29110 : *activité d'architecture et conception, construction logicielle, intégration et test, et l'activité de livraison de produit*; comme compatibles aux F/OSS. Nous pouvons ainsi déduire des résultats de compatibilité de la norme ISO/IEC 29110 aux F/OSS (chapitre 6 – tableau 6.12), que **la norme ISO/IEC 29110 est compatible à 67% aux F/OSS**, car 4 sur 6 des activités de développement prescrits par la norme, sont compatibles aux processus libres. Cependant, la seule connaissance des activités de ISO/IEC 29110 compatibles aux F/OSS, n'est pas suffisante pour pallier aux lacunes de qualité observées dans les F/OSS, ni pour améliorer même en partie les processus libres et donc, la qualité des produits logiciels libres. Ainsi, la question qui est dès lors soulevée est de voir comment appliquer la norme ISO/IEC 29110 aux processus de développement libres pour aider à les améliorer, compte tenu de la culture, des valeurs et de la philosophie des F/OSS.

Compte tenu de notre analyse des résultats obtenus au chapitre 6, les processus d'analyse des exigences, d'architecture et de conception et le processus d'intégration et tests du processus de développement libre, comportent la majorité des points d'écarts par rapport au développement classique de logiciels tel que prescrit par la norme ISO/IEC 29110. Et les activités de la norme ISO/IEC 29110 : activité d'architecture et conception, construction logicielle, intégration et test, et l'activité de

livraison de produit; ont été identifiées compatibles aux F/OSS. Une proposition d'intervention à court terme pour l'amélioration des performances du processus de développement des logiciels libres serait, d'appliquer initialement la norme ISO/IEC 29110 uniquement sur le processus d'architecture et conception, et le processus d'intégration et tests du processus de développement libre. Ce choix est fait, car ces processus sont à la fois des activités de la norme ISO/IEC 29110 compatibles aux F/OSS et également des processus du processus de développement libre présentant la majorité des points d'écarts avec le développement classique de logiciel, selon les prescriptions de la norme ISO/IEC 29110. La proposition d'intervention à long terme pour augmenter les performances du processus de développement des F/OSS serait donc, d'appliquer la norme ISO/IEC 29110 sur le reste de ces activités qui sont compatibles aux F/OSS.

Dans la suite, nous ferons des discussions quant aux différents aspects de notre application de la norme ISO/IEC 29110 au contexte particulier des logiciels libres.

7.3.1 Approche d'application de la norme aux logiciels libres

Compte tenu du contexte des F/OSS, nous optons pour une approche analytique, axée sur l'aspect social de leur développement. La motivation de notre approche est soutenue par les valeurs des F/OSS et les quatre libertés de la philosophie du libre.

Toutefois, sachant que la norme ISO/IEC 29110 prescrit les processus minimaux que doivent respecter une organisation développant du logiciel. Par conséquent, nous pouvons être porté à croire que si quelques uns des activités prescrites par la norme ISO/IEC 29110 ne sont pas compatibles aux F/OSS pour une raison ou une autre, la norme pourrait ne pas être appliquée au contexte libre. Cependant, le but que nous recherchons n'est pas de changer le développement libre, mais d'améliorer en partie le processus de développement libre. Aussi, la présence de ces deux activités

incompatibles aux F/OSS ne constitue pas un frein ou un obstacle à notre approche si le reste des activités prescrites par la norme peuvent contribuer à améliorer les performances en termes de qualité du processus de développement libre. De plus, dans le contexte du libre, l'idée n'est pas de rendre leur développement aussi rigoureux et procédural que celui en entreprise, mais d'améliorer leur processus de développement et donc, la qualité des produits logiciels libres, tout en respectant et en conservant cette liberté qu'ont les participants dans les projets libres. Ainsi, sachant que deux des activités du processus d'implémentation de logiciels de la norme ISO/IEC 29110 sont incompatibles aux F/OSS, nous pouvons tout de même poursuivre notre application de la norme aux F/OSS. Au contraire, si l'es activité d'initiation d'implémentation et l'activité d'analyse des exigences de la norme s'étaient avérées compatibles aux F/OSS suite à notre analyse, nous aurions eu de sérieuses difficultés à les accorder aux processus libres compte tenu de leur philosophie et de leur culture. L'application des prescriptions de la norme lors du déroulement de ces processus libres, impliquerait de complètement changer la façon de procéder des F/OSS, chose qui selon nous, ne sera pas envisageable dans le contexte libre.

La particularité et l'avantage de notre approche est qu'elle se conforme aux valeurs et à la culture des F/OSS, compte tenu de leur aspect social (voir type de développement : chapitre 3 - section 3.1.4.1). Nous ne prenons pas en considération le nombre de participants au projet de F/OSS car les participants aux projets libres, bien que nombreux sont répartis en de multiples sous projets, eux-mêmes subdivisés en de plus petits sous-projets. Nous proposons pour une parfaite conformité à la norme que le développement de F/OSS soit constitué en plusieurs pools de développeurs dont le nombre soit inferieur ou égal à 25. Les gestionnaires de projets de F/OSS pour chaque module du projet devront s'assurer que le projet qu'ils maintiennent ne contient pas plus de 25 personnes par développement de composant. De ce fait, la norme ISO/IEC 29110 pourra être appliquée au développement libre car les projets de

F/OSS respecteront dès lors le cadre conceptuel de celle-ci. Par ailleurs, comme autre considération de notre approche de l'application de la norme aux F/OSS, nous avons choisi de supprimer dans certaines activités prescrites par la norme, la tâche consistant à assigner les tâches aux membres de l'équipe de travail selon leur rôle. La suppression de cette tâche dans notre approche d'application de la norme au développement libre, permettra de rendre notre approche plus flexible et fera en sorte qu'elle s'accorde bien au développement libre. Notre approche permettra d'améliorer le processus de développement libre par l'ajout de plus de documentation, qui va permettre aux F/OSS non seulement d'élargir leur communauté en attirant de plus en plus d'utilisateurs et d'entreprises commerciales, mais aussi de garantir une qualité élevée pour tout produit logiciel développé ou maintenu.

Les différents aspects que nous considérons pour notre approche d'application de la norme ISO/IEC 29110 au contexte du libre, sont les suivants :

➢ **Les artefacts :**

Seuls les artefacts en sortie nous intéressent, car ils représentent les preuves de la réalisation d'une tâche pour un processus donné. Concernant les activités de la norme compatibles aux F/OSS, nous retenons pour les F/OSS, les artefacts suivants :

- **Architecture et conception :** les artefacts que nous retenons dans le sens des F/OSS sont les suivants :
    - Document de spécification des exigences
    - Document de conception du logiciel
    - Rapport de traçabilité
    - Les cas et les procédures de tests

Sachant que les F/OSS souffrent d'un manque apparent de documentation, ces artefacts constitueront une base solide de documentation pour le processus d'architecture et de conception. Dans les industries telles que le veulent les bonnes pratiques du génie logiciel, les exigences du logiciel doivent avoir été clairement définies et de manière précises, afin de mener à bien ce processus d'architecture et de conception. Ainsi, une bonne compréhension des spécifications du logiciel est nécessaire pour définir ou pour mettre à jour la conception du logiciel. De plus, réaliser de bonnes architectures et de bonnes conceptions logicielles, permettent aux développeurs de produire ultérieurement du code de qualité. Cependant, nous savons de par l'observation des F/OSS (chapitre 3 et 5) qu'ils produisent déjà du logiciel de qualité. De plus, le processus d'architecture et de conception est implicite à tout développement dans le contexte libre [35, 79, 94, 104]. De ce fait, avoir un document complet et détaillé de conception logicielle, serait pour les F/OSS, une tâche très difficile à réaliser.

Exiger la réalisation d'un tel document de conception dans les F/OSS semblerait inutile à premier abord, puisqu'ils produisent déjà du code de qualité. D'un autre côté, la présence d'un document formel de conception dans le développement libre, permettrait aux F/OSS de mieux gérer et de mieux structurer leur développement. Cela garantira la production d'un code de qualité élevée pour tout projet libre développé ou maintenu dans n'importe quelle communauté libre. De plus, un tel document de conception logicielle, permettra également aux F/OSS d'accroitre leur communauté en attirant de plus en plus de participants techniques compte tenu de la disponibilité des ressources libres ainsi que de la transparence des processus libres. En ce qui concerne l'attraction des participants et des utilisateurs non techniques aux projets libres, elle nécessite essentiellement que les exigences du logiciel, prenant forme à travers les discussions entre développeurs sur les *mailing lists*, soient respectées et surtout opérationnelles. Les utilisateurs des F/OSS pourront toujours participer à la réalisation de ce document en fournissant des feedbacks, par exemple

des suggestions, comme cela se fait déjà dans les F/OSS pour permettre de détecter et de corriger le plus rapidement possible les erreurs s'y trouvant, et ainsi permettre d'aboutir à une consistance du document de conception.

Afin d'avoir un tel document de conception logicielle, nous proposons comme solution que l'un des membres du groupe de mainteneurs du projet ou développeurs de confiance du projet, ayant une assez bonne connaissance de la conception courante du logiciel, se charge de la réalisation et de la documentation de la conception logicielle; ceci à l'opposé des programmeurs (qui documentent la conception logicielle, tâche qui n'est cependant pas obligatoire) comme cela se fait habituellement dans les F/OSS. Et afin de vérifier la consistance du document de conception de logiciel, il peut annoncer une revue par les pairs sur la liste de diffusion de développeurs pour obtenir des feedbacks usuels. De cette manière, les utilisateurs ou autres développeurs pourront influencer les décisions. Les F/OSS gagneraient énormément à adopter notre proposition, d'autant plus qu'elle est conforme à l'un des principes du développement libre, qui consiste à traiter les utilisateurs comme co-développeurs pour favoriser l'avancement rapide du développement et la qualité du produit logiciel qui en résultera [102]. Par ailleurs, la communauté Mozilla fonctionne d'une manière presque semblable à notre proposition. Dans Mozilla, ce sont le propriétaire du module et le super examinateur (rappel chapitre 5), possédant une assez bonne compréhension de la conception courante, qui sont responsables de mettre à jour la conception du logiciel [104].

Quant au rapport de traçabilité, la norme ISO/IEC 29110 prescrit qu'il devra incorporer les relations entre les exigences et les éléments de conception logicielle, les cas et les procédures de tests. En ce qui concerne le document de cas de tests (rapport de test) [15], document de conception logicielle, document de traçabilité, document de requêtes de changements [104], Mozilla les utilise déjà.

- **Construction logicielle :** les artefacts retenus sont les suivants :
    - Standards d'écriture de code : cette documentation permettrait pour chaque communauté de F/OSS d'avoir une unique façon d'écrire le code, chose qui permettrait d'accélérer le contrôle du code source, et donc, permettrait de retenir plus de contributions importantes. Tel qu'identifié par Michlmayr [91], certains F/OSS à grand nombre de contributeurs, définissent des *documentations de style de codage* et *des documentations de soumissions de code*. Le résultat auquel nous voulons aboutir à ce niveau est l'usage récurrent des standards d'écriture de code comme pratique de qualité dans les F/OSS.
    - Cas de test unitaires : dans les F/OSS, chaque programmeur réalisant une fonctionnalité du logiciel, effectue déjà des cas de tests unitaires afin de s'assurer de la bonne marche de cette fonctionnalité. C'est une pratique très courante dans les projets libres.
    - Rapport de traçabilité : ce rapport devrait permettre pour les F/OSS, de facilement repérer les composants logiciels qui ont été construits ou modifiés.

- **Intégration et tests :** l'application de la norme portera essentiellement sur le **rapport de tests** et le **rapport de traçabilité**.
    - Rapport de tests : il va consister à documenter les résultats des tests ayant été opérés pour l'intégration du logiciel.
    - Rapport de traçabilité : le document de traçabilité sera défini s'il n'existe pas encore. Dans le cas contraire, il sera mis à jour par l'ajout de l'intégration des différentes relations entre les composants logiciels ayant été construits ou modifiés.

Tel que nous l'avons identifié par notre évaluation du processus de développement libre, ces documents n'existent habituellement pas dans les F/OSS. Cependant, nous

avons remarqué qu'ils pourraient être reconstruits à partir des archives du CVS sur les composants du logiciel pour le rapport de traçabilité, et à partir des discussions de traitement de bogues se trouvant sur les listes de diffusion des développeurs pour le rapport de tests. Un bogue dans les F/OSS peut être échangé comme le code entre les développeurs et est associé à certaines caractéristiques : un numéro d'identification, le nom de l'auteur/propriétaire, les fichiers qui lui sont attachés, la gravité et priorité du bogue, le statut et enfin le nombre de commentaires [104]. Quant au CVS, les archives qui y sont incluses, contiennent habituellement des informations sur le nom, la version, le mainteneur, la licence, la taille et autres informations sur le *package* ou composant logiciel, le plus important étant que, chaque *package* contient des références[35] vers d'autres *packages* nécessaires à l'exécution [113].

Bien que les défauts du logiciel dans les F/OSS soient détectés et corrigés par de nombreux individus [102] bien avant la phase d'intégration [26], le suivi des prescriptions de la norme ISO/IEC 29110 par l'approche proposée, permettra aux F/OSS d'améliorer davantage la qualité de ce processus libre, tout en conservant tant les libertés des utilisateurs que la transparence des documents et celle du processus. Les tâches de coordination et d'intégration des toutes les contributions des développeurs d'un module dans les F/OSS sont faites par les chefs ou mainteneurs ou gestionnaires de ces modules [59, 94, 104] (voir chapitre 5 pour les détails de ce processus dans les F/OSS). Certains projets libres appliquent à ce niveau une construction automatique du système supportée par des outils tels que : Tinderbox (cas Mozilla [104]), DejaGnu (cas Linux [49]). Étant donné qu'un module du projet dans les F/OSS est subdivisé en sous-compartiments, généralement relatif au développement de ses composants logiciels, et que nous avons choisi de diviser le développement dans les F/OSS en pools dont le nombre de développeurs est inférieur

---

[35] Les références dont il est questions consistent en des informations de dépendances portant les statuts suivants : « requis », « recommandé » ou « suggéré » [36].

ou égale à 25 par composants logiciels pour que les F/OSS respectent plus le cadre contextuel de la norme ISO/IEC 29110, ainsi chaque compartiment ou composant d'un module sera constitué d'au plus 25 développeurs, qui seront pour ces composants les mainteneurs. Et donc, ils coordonneront et intégreront toutes les contributions des développeurs externes à leur module [59, 83, 94, 104]. De plus nous recommandons qu'en plus des tâches qui leur sont attribuées dans leur communauté libre, ces mainteneurs étant à la base des développeurs, devront se charger de rédiger le rapport de tests et le rapport de traçabilité pour l'intégration de leur composant. Ils pourront procéder par vote entre leurs membres pour désigner la personne responsable de cette tâche. Une fois ces documents rédigés, ils seront soumis aux autres développeurs pour révision. Ainsi, tout utilisateur voulant participer à ce processus peut le faire en soumettant des suggestions pour la vérification de ces artefacts ou en réalisant les tests. De cette manière, notre proposition, conservera non seulement les bonnes pratiques du développement des F/OSS, mais également améliorera ce dernier par l'ajout des documents de traçabilité et rapport de tests d'intégration dont souffrent actuellement les F/OSS d'après notre évaluation (voir chapitre 6). L'intégration du système logiciel libre tout entier se poursuivra jusqu'à la couche supérieure des mainteneurs du projet.

- **Livraison de produit :** les différents artefacts retenus pour notre personnalisation à ce niveau sont :
    - Documentation de maintenance
    - Configuration logicielle

Ce paragraphe clôt ce premier aspect de notre application de la norme ISO/IEC 29110 aux F/OSS – *aspect des artefacts*. Nous savons déjà par les travaux de Michlmayr et ses collaborateurs [92] et d'après les résultats de notre évaluation du processus de développement des F/OSS (confer chapitre 6 – section 6.3), qu'ils souffrent d'un manque apparent de documentation. Selon notre observation, la raison principale de

ce problème de documentation du processus de développement de logiciels libres est que les F/OSS sont conduits par des bénévoles, et donc, la tâche de documenter un processus n'est pas une tâche obligatoire. Afin de pallier à cette lacune de qualité, nous proposons donc dans chaque processus de développement libre, que des **listes de contrôles de documents ou des artefacts** soient appliquées dans les processus libres pour qu'ils soient conformes aux prescriptions de la norme ISO/IEC 29110. Cette proposition s'ajoute aux différents artefacts relatifs à notre application de la norme ISO/IEC 29110 aux F/OSS, présentés dans les paragraphes précédents. De plus, nous savons déjà que certains F/OSS appliquent des listes de contrôle de tests [91] comme pratique de qualité.

- **Les rôles :**

Les différents rôles prescrits par la norme seront retenus pour les F/OSS. Cependant, compte tenu du contexte libre, le fait que les participants au projet soient essentiellement des bénévoles, nous ne pouvons recommander le suivi strict des rôles prescrits par la norme, et ce, pour chacune des activités de la norme ISO/IEC 29110 compatibles aux F/OSS. Ces participants aux projets libres, bénévoles et experts en leur domaine, jouent des rôles dans le développement des logiciels libres, pour chacun des différents processus de développement. Les F/OSS comme tout projet de développement de logiciels, se focalisent sur l'implémentation du logiciel.

Nous savons déjà que certains F/OSS développent du logiciel de qualité élevée et maintiennent ce niveau de qualité pour tous les logiciels développés ou maintenus dans ces communautés. L'objectif que nous souhaitons atteindre à ce niveau est de permettre à tout projet libre dans n'importe quelle communauté de F/OSS, d'aboutir **à coup sûre** à un produit logiciel qui soit de qualité élevée et de **maintenir** ce niveau de qualité pour tout produit logiciel développé ou maintenu. Le pré-requis essentiel à

cet effet, est bien sûr que le nombre de participants qui maintiennent le projet libre soit au minimum de 15 [26]. Nous avons choisi de limiter ce nombre au maximum à 25 développeurs par composant d'un module du projet pour que les F/OSS soient conformes à la norme ISO/IEC 29110. De plus, nous savons que les participants aux projets libres jouent habituellement plusieurs rôles à la fois dans le déroulement du processus de développement du logiciel. L'essentiel étant la réalisation du projet et d'arriver rapidement en une version stable du logiciel qu'il faudra aussitôt livrer.

- **Architecture et conception :** la norme prescrit trois rôles qui sont : chef technique (*Technical lead*), analyste et concepteur (*Designer*). Dans les F/OSS compte tenu des résultats de notre évaluation de leur processus de développement (confer chapitre 6 – section 6.3), nous allons retenir les rôles de clef technique et programmeur. Cependant, comme nous le savons déjà de notre analyse du développement libre, les programmeurs jouent à ce niveau les rôles d'analyste et de concepteur.

- **Construction logicielle:** la norme prescrit deux rôles qui sont : clef technique et programmeur. Ces rôles sont parfaitement respectés dans les F/OSS d'après notre évaluation du processus de développement libre (confer chapitre 6 – section 6.3).

- **Intégration et tests :** la norme prescrit quatre rôles qui sont : chef technique, programmeur, client et analyste. Selon les résultats de notre évaluation du processus de développement des logiciels libres (confer chapitre 6 – section 6.3), les programmeurs portent aussi le chapeau d'analyste et souvent celui de client. Les clients étant les utilisateurs du logiciel libre. Par utilisateurs, nous entendons, de simples individus ou alors des programmeurs, ou encore des entreprises.

- **Livraison de produit :** la norme prescrit trois rôles qui sont : chef technique, équipe de travail et concepteur. Dans les F/OSS, nous savons par l'observation de leurs pratiques que seuls les programmeurs et les mainteneurs de projets participent à la réalisation de ce processus (confer chapitre 6 – section 6.3). Le nombre des mainteneurs de projets est au moins de 15 [26], et nous l'avons limité au maximum à 25. De plus, il est connu dans les F/OSS que chaque participant aux projets libres, choisit la tâche à réaliser. Cela pourrait ainsi nous amener à penser que dans les F/OSS, il n'y a aucune répartition des rôles. En considérant cette idée, un participant peut avoir une multitude de rôles. Les développeurs externes au groupe de développeurs de confiance peuvent également participer à la réalisation de cette activité. Pour appliquer la norme à ce niveau au sens des F/OSS, nous ne retenons que les deux rôles suivants : chef technique et programmeur (ce dernier se comportera pour le processus de livraison de produit comme un designer).

➢ **Les tâches :**

De par la création même du concept de logiciel libre et face aux rigueurs procédurales du développement commercial, les programmeurs ou les participants aux F/OSS ont voulu une certaine liberté concernant la disponibilité du code (pour les 4 libertés du libre [101]) et le choix des tâches à réaliser. De ce fait, les F/OSS sont non conformes au développement commercial, et en particulier dans le cadre des prescriptions de la norme ISO/IEC 29110, à la tâche d'affectation du travail aux membres de l'équipe. À cet effet, pour notre approche de l'application de la norme ISO/IEC 29110 aux F/OSS, nous avons choisi de supprimer *la tâche d'affectation des tâches aux membres de l'équipe de travail selon leur rôle*, question de rendre notre application de la norme plus flexible et d'être facilement applicable aux F/OSS. Hormis cette tâche, nous trouvons que les autres tâches prescrites par la norme, notamment pour

les activités de la norme qui sont compatibles aux F/OSS, peuvent être suivies dans le processus de développement des F/OSS.

En tenant compte de l'aspect de la structure informelle des F/OSS, ainsi que de l'ouverture étonnante (faisant référence au style de développement ouvert [92] ainsi qu'à la disponibilité des ressources) des projets libres, les tâches de documentation seront plutôt difficiles à respecter dans les F/OSS. Cela conduit ainsi à un manque de documentation dans certains projets libres. Il est cependant possible de retrouver dans d'autres projets libres la documentation du système en ligne ou la documentation destinée à être imprimée pour le support des utilisateurs finaux ou des développeurs [104, 106]. Cela dit, certains des travaux précédents [92], de même que notre analyse du processus de développement libre, ont eu à identifier cette lacune de qualité des F/OSS. Face à cela, nous recommandons que les mainteneurs de projets dans les communautés de F/OSS se chargent de ces tâches. Nous proposons également afin de s'assurer que les différentes documentations ont bel et bien été faites :

- Que les membres du groupe de mainteneurs de projets pour un module donné, élisent une personne responsable de la documentation de processus. Exception faite de Mozilla pour qui chaque composant du module du projet, l'auteur du composant est également le propriétaire du module et il représente l'autorité de prise de décision concernant ce module [104]. Donc dans ce cas précis, en plus des tâches qui lui sont allouées dans la communauté, il devra définir ou mettre à jour les différentes documentations relatives à son module;
- Et une fois la documentation faite, que la personne retenue envoie la documentation réalisée aux autres membres pour révision. Dans le cas échéant, l'un des membres du groupe de mainteneurs, désigné par ses confrères, sera chargé de vérifier que les différents documents relatifs à un processus donné soient bel et bien définis. À ce niveau dans le cas de Mozilla, le propriétaire du module peut travailler avec un membre de l'équipe d'assurance qualité pour vérifier la consistance des différentes

documentations dans chacun des processus de développement avec le composant développé ou maintenu [104].

Cette proposition sera plutôt conforme aux pratiques des F/OSS et constituera en elle-même une manière de s'assurer que la documentation soit effectivement faite, tout au contraire de laisser la tâche de documentation optionnelle entre le groupe de développeurs de confiance ou mainteneurs de projet. Cependant, notre proposition de solution au problème de documentation du processus de développement des F/OSS ne vise pas à interdire aux participants aux F/OSS de contribuer en réalisant les tâches de documentation ou par des suggestions.

7.4 CONCLUSION

Lors de notre observation et analyse du processus de développement de logiciels libres, nous avons pu constater que les activités prescrites par la norme ISO/IEC 29110 sont réalisées dans les F/OSS, avec cependant certaines différences notables. Nous avons pu identifier au chapitre 6, les activités de la norme ISO/IEC 29110 qui sont compatibles et celles non compatibles à la culture et aux valeurs des F/OSS. Ainsi, dans ce chapitre, nous n'avons étudié en détails que les activités de la norme qui sont compatibles aux F/OSS, et avons ainsi pu identifier les bénéfices potentiels de la norme pour les F/OSS. Cet aspect de compatibilité de la norme ISO/IEC 29110 aux F/OSS, nous a ainsi permis d'introduire le problème de l'application de la norme ISO/IEC 29110 aux F/OSS. Sachant que seuls 4 processus de la norme sont compatibles aux F/OSS, notre approche d'application de la norme ISO/IEC 29110 aux F/OSS est donc restreinte à ces processus dans le développement libre. Notre approche tourne autour des artefacts, des rôles et des tâches relatives à chacun des processus respectifs de développement des F/OSS : architecture et conception, construction logicielle, intégration et tests, et pour finir livraison de produit. Pour

chaque aspect (artefacts, rôles, tâches) de notre approche, nous avons effectué une réflexion poussée pour chaque processus libre. Nous avons également proposé dans certains cas des idées qui permettraient aux F/OSS d'améliorer leur processus en termes de documentation, de même que des idées qui aideront à rendre leur développement un peu plus structuré et organisé tout en conservant les libertés des utilisateurs. Pour se faire, la rigueur du développement sera exigée aux mainteneurs du projet, particulièrement concernant les tâches de documentation. Cependant, notre approche n'interdit pas aux participants externes au groupe de mainteneurs du projet dans les F/OSS, de contribuer au projet par des tâches de documentation ou par des suggestions.

CHAPITRE VIII

CONCLUSION

Le thème de ce mémoire est un thème très intéressant, d'emblé parce que le domaine du logiciel libre est un domaine d'actualité qui intéresse de nombreux chercheurs, de simples individus utilisateurs de logiciels libres, de même que des entreprises. Le domaine du libre regorge de nombreuses problématiques demeurant encore ouvertes, notamment dans le cadre de la qualité. D'autre part, le sujet de ce mémoire suscite un réel intérêt, typiquement en ce qui concerne la question de l'application des normes d'ingénierie tout au long du processus de développement de logiciel libre.

Ainsi, le présent travail est principalement analytique et conclut une année et deux mois de recherche sur les logiciels libres. Il se concentre sur la problématique générale de qualité dans le développement des F/OSS et particulièrement, sur la question de savoir si la norme ISO/IEC 29110 peut soutenir et améliorer le processus de développement de logiciels libres compte tenu de leurs valeurs et de leur culture.

Afin de mener à bien ce travail de recherche, quatre objectifs de recherche ont été fixés au chapitre 2. Le premier objectif consistait à identifier les problèmes de qualité que rencontrent les projets de F/OSS. Afin d'atteindre cet objectif, nous avons dû nous familiariser aux F/OSS dans le chapitre 3. Ainsi, l'observation initiale que nous avons faite sur les F/OSS nous a permis de mieux comprendre leur fonctionnement et

leur culture. Par ailleurs, cette observation des F/OSS, a également permis de faire ressortir les valeurs des F/OSS, d'identifier les principes soutenant chacune des communautés Linux, Apache et Mozilla. De plus, elle nous a permis d'identifier certaines pratiques de qualité des F/OSS.

Dans un cadre méthodologique, le second objectif de notre recherche, axé sur l'évaluation qualitative du processus de développement des F/OSS, a consisté à analyser en profondeur le processus de développement de logiciel libre en vue d'identifier les différences entre le développement libre et le développement classique de logiciel; et d'identifier les activités de la norme ISO/IEC 29110 qui sont compatibles et celles non compatibles aux F/OSS. Nous avons ainsi évalué le processus de développement de logiciel libre selon les prescriptions de la norme ISO/IEC 29110, et inversement.

Cet objectif a permis nettement dans le chapitre 6, d'identifier les différences entre le développement libre et le développement classique de logiciels. Ce résultat constitue la première partie de notre troisième objectif de recherche. Ainsi, nous avons remarqué que, la majorité des points d'écarts entre le processus de développement de F/OSS et le développement classique de logiciels se situe au niveau des activités d'analyse des exigences, d'architecture et de conception, et l'activité d'intégration et de tests. Ces écarts sont de deux types :

- Les écarts de documentation. Ils sont les plus importants. Ils sont principalement causés par le bénévolat des participants aux projets de F/OSS. Par ailleurs, une autre raison pourrait être l'absence d'une étape de vérification et de validation de documents;
- Les écarts de rôles. Ces derniers sont moins importants car même si les tâches d'un des processus du développement libre sont réalisées par une seule personne, cette personne porte plusieurs chapeaux. Et donc, les prescriptions de la norme ISO/IEC 29110 sont respectées.

Par ailleurs, ce résultat a contribué à identifier à la fois les différences générales entre le processus de développement libre et le processus de développement en ingénierie, et plus en détail, les différences entre chaque activité des différents processus de développement, aux chapitres 6 et 7.

Afin d'atteindre le second objectif de notre recherche, nous avons dû présenter de manière détaillée la philosophie des F/OSS dans le chapitre 5. Le second objectif de notre recherche a procuré comme avantage l'introduction d'une stratégie d'amélioration du processus de développement libre, basée sur l'application de la norme ISO/IEC 29110 aux F/OSS. Le point de départ de cette stratégie d'amélioration du processus de développement libre a été une étude de la compatibilité de la norme ISO/IEC 29110 à la culture, aux valeurs et à la philosophie des F/OSS. La particularité de notre approche pour étudier la compatibilité de la norme ISO/IEC 29110 aux F/OSS est qu'elle est basée sur certains critères de compatibilité aux F/OSS que nous avons eu à identifier par une étude rigoureuse des sous-projets Noyau Linux, GCC, Firefox, GNOME et Apache http Server. Les critères de compatibilité identifiés des F/OSS, ont été la base de notre étude de la compatibilité de la norme ISO/IEC 29110 aux F/OSS. Elle a eu pour avantage l'identification des activités de la norme ISO/IEC 29110 qui sont compatibles aux F/OSS, dans le chapitre 6. Ainsi, ce résultat constitue la seconde partie du troisième objectif de notre recherche. Nous avons donc clairement identifié les activités : d'architecture et conception, de construction logicielle, d'intégration et tests, et enfin de livraison du produit; comme étant compatibles aux F/OSS. Par la suite au chapitre 7, nous avons étudié en profondeur ces quatre activités et nous avons proposé une façon d'appliquer la norme ISO/IEC 29110 aux F/OSS, tout en se conformant aux valeurs et à la culture des F/OSS. Cette approche de l'application de la norme ISO/IEC 29110 aux F/OSS constitue un avantage important de ce mémoire. Elle a

conduit à certaines améliorations du processus de développement des F/OSS. Ces améliorations seront vues dans le paragraphe suivant.

Comme précédemment vu dans ce mémoire, nous savons que les projets de F/OSS sont habituellement non structurés, mais produisent dans certains cas des logiciels d'une qualité élevée qui est quelque fois comparable, voire supérieure à celle des logiciels développés en entreprise. Et donc, le quatrième objectif de notre mémoire, a permis de faire ressortir au chapitre 7 que : **l'usage de la norme ISO/IEC 29110 permettra selon l'étude menée dans ce mémoire, d'améliorer le processus de développement des logiciels en augmentant grandement ou en garantissant la documentation des projets libres**. L'ajout de plus de documentation au processus de développement libre va permettre aux F/OSS selon nous, d'élargir leurs communautés en attirant de plus en plus d'utilisateurs et d'entreprises commerciales. D'autre part, l'usage de la norme ISO/IEC 29110 dans les processus libres, permettrait aux F/OSS d'avoir une méthodologie de développement un peu plus organisée. Ainsi, cela va permettre à toute communauté de F/OSS de garantir, de maintenir ou d'améliorer ce niveau de qualité élevée, déjà présent dans certains produits logiciels libres. De ces résultats, nous sommes donc convaincu que la norme ISO/IEC 29110 appliquée dans un environnement de développement libre, **va contribuer à améliorer de manière significative** la qualité du processus de développement libre. Ainsi donc, **elle peut soutenir le processus de développement des F/OSS compte tenu de leurs valeurs et de leur culture**. Ce qui répond à la question de cette recherche, laquelle a été posée au chapitre 2.

Nous estimons donc que les objectifs que nous nous sommes fixés pour ce travail de recherche ont été pleinement atteints.

Les **contributions** de ce mémoire sont nombreuses. Nous les résumons donc comme suit : nous citons premièrement, les particularités de chacun des projets Linux,

Apache, Mozilla et GNOME; de mêmes que les similitudes entre ces projets. La seconde contribution de ce mémoire est l'identification des différences entre le processus de développement de logiciels libres et le processus classique de développement de logiciels. Une autre contribution de ce mémoire se situe à l'introduction du concept de compatibilité aux valeurs, à la culture et à la philosophie des F/OSS, d'une norme de base sur le développement, dans le cadre de ce mémoire : la ISO/IEC 29110. Ce concept a permis l'identification de certains critères de compatibilité aux F/OSS, faisant ainsi le pont entre la norme ISO/IEC 29110 et notre proposition d'application de cette norme aux F/OSS. Notre approche d'application de la norme aux F/OSS, nous a donc permis d'identifier les améliorations que procurera la norme au processus de développement des F/OSS. Les critères de compatibilité aux F/OSS que nous avons identifiés, sont en eux-mêmes une des contributions importante de ce rapport de recherche. De plus, comme autre contribution de ce mémoire, nous avons identifié les différentes activités de la norme ISO/IEC 29110 qui sont compatibles aux F/OSS, et donc, qui pourraient s'appliquer aux F/OSS. Pour finir, nous citons également notre approche d'application de la norme ISO/IEC 29110 au contexte des F/OSS comme autre contribution de ce mémoire.

Nous ne saurons cependant oublier de noter que sur le plan personnel, ce travail nous a permis de nous initier à la recherche. Le thème de cette recherche a été plus profitable que ce à quoi nous nous attendions, car nous avons acquis de nombreuses connaissances sur les F/OSS.

L'approche utilisée dans ce travail pour analyser et améliorer les performances du processus de développement des F/OSS, n'est pas exclusive. Elle constitue une approche parmi tant d'autres. Le travail présenté dans ce mémoire n'aborde pas les aspects d'application des méthodes de vérification et de validation du génie logiciel aux processus de développement de logiciels libres question de forcer les communautés de F/OSS à être conformes aux normes. Cette idée peut constituer une

continuité à ce travail de recherche. Également, le travail que nous présentons ne comptait pas parmi ses objectifs de valider nos suggestions d'amélioration du processus de développement libre. Nous sommes conscients que certaines de nos suggestions pourraient être reçues avec une certaine réticence par les communautés libres. Ainsi comme travail futur, nous pourrions effectuer une phase de validation des nos suggestions d'améliorations du processus de développement libre.

Par ailleurs, dans une perspective de recherche plus étendue, ce mémoire pourrait constituer une base de recherche pour l'application des normes de développement du génie logiciel aux F/OSS par le biais des critères de compatibilité aux F/OSS ayant été identifiés dans ce rapport. D'autre part, afin d'étendre davantage la recherche, nous pourrions éventuellement songer à élargir le nombre de critères de compatibilité aux F/OSS en faisant une enquête auprès des membres des projets libres étudiés dans ce mémoire et en scrutant minutieusement les archives de ces projets. Cette tâche ne serait pas très aisée car il nous faudra pour cela : soit rencontrer ou simplement être en contact de manière régulière avec plus d'une centaine de personnes par projet libre pour parfaire et/ou agrandir les bases d'informations relatives aux projets libres retenus dans ce mémoire. Ce qui nécessiterait un effort considérable pour recueillir ces informations. Par la suite, ces informations seraient combinées à celles déjà acquises sur les F/OSS. L'aboutissement de ce travail permettra d'avoir un ensemble complet de critères de compatibilité aux F/OSS qui pourrait donc éventuellement servir de modèle de mesure pour l'évaluation de la compatibilité de n'importe quelle norme de base sur le développement de logiciel conformément à la culture, aux valeurs et à la philosophie des F/OSS. Par ailleurs, une autre extension de ce travail de recherche consisterait à faire le suivi des communautés libres appliquant la nouvelle norme ISO/IEC 29110 sur leur processus de développement afin de valider les suggestions d'améliorations faites au cours de notre approche d'application de la norme ISO/IEC 29110 au processus de développement libre. Dans l'affirmative, nous aurions alors prouvé que la norme ISO/IEC 29110 peut améliorer le processus de

développement libre et rendrait crédible son application dans un environnement libre. De plus, cela soutiendrait également que des normes de base sur le développement peuvent s'appliquer au contexte libre.

Nous espérons que ce travail sera profitable aux projets de F/OSS et aidera à prolonger la recherche dans les F/OSS.

ANNEXES

NB : Les références utilisées dans les annexes A, B et C sont présentes dans la bibliographie du mémoire. Les références utilisées dans les annexes D et E se situent juste à la suite du contenu de chacune des annexes D et E car le texte des annexes D et E a été recopié tel quel.

ANNEXE A : PRINCIPES DE LOGICIELS OPEN SOURCE

Pour qu'un logiciel soit considéré comme un logiciel « Open Source », il faut qu'il soit conforme aux critères suivants, définis par OSI [23] :

La libre redistribution

La licence ne doit pas restreindre une partie de vendre ou de donner le logiciel en tant que composante d'un groupe de distribution de logiciels contenant des programmes de plusieurs sources différentes. La licence ne doit pas exiger une redevance ou d'autres choses, pour une telle vente.

Le code source

Le programme doit comprendre le code source et doit permettre une répartition en code source ainsi que la forme compilée. Si une certaine forme d'un produit n'est pas distribuée avec le code source, il doit y avoir un moyen bien connu de l'obtention du code source pour plus d'un coût raisonnable, le téléchargement via Internet sans frais. Le code source doit être la forme privilégiée dans laquelle un programmeur modifie le programme. Délibérément occulté le code source n'est pas autorisé. Des formes intermédiaires comme la sortie d'un préprocesseur ou traducteur, ne sont pas autorisés.

Les travaux dérivés

La licence doit autoriser les modifications et les travaux dérivés, et doit leur permettre d'être distribués sous les mêmes conditions que la licence du logiciel original.

L'intégrité du code source des auteurs

La licence peut restreindre le code source d'être distribué sous une forme modifiée, seulement si la licence permet la distribution de fichiers « patch » avec le code source dans le but de modifier le programme au moment de la construction. La licence doit explicitement permettre la distribution de logiciels compilés à partir du code source modifié. La licence peut exiger les travaux dérivés pour porter un nom différent ou un numéro de version du logiciel d'origine.

Pas de discrimination contre les personnes et les groupes de personnes

La licence ne doit pas faire de discrimination contre toute personne ou groupe de personnes.

Pas de discrimination contre les champs d'effort

La licence ne doit pas restreindre toute personne de faire usage de ce programme dans un domaine spécifique de l'activité. Par exemple, elle ne peut pas restreindre le programme d'être utilisé dans une entreprise, ou d'être utilisé pour la recherche génétique.

Distribution de la licence

Les droits attachés au programme doivent s'appliquer à tous ceux à qui le programme est redistribué, sans qu'il ne soit nécessaire pour une exécution d'une licence supplémentaire par ces parties.

La licence ne doit pas être spécifique à un produit

Les droits attachés au programme ne doivent pas dépendre d'une partie de programme d'une distribution de logiciels en particulier. Si le programme est extrait à partir de cette distribution et utilisé ou distribué selon les termes de la licence du programme, toutes les parties à qui le programme est redistribué doivent avoir les mêmes droits que ceux qui sont accordés en liaison avec la distribution de logiciels originaux.

La licence ne doit pas restreindre d'autres logiciels

La licence ne doit pas imposer des restrictions sur d'autres logiciels qui sont distribués avec le logiciel sous licence. Par exemple, la licence ne doit pas insister pour que tous les autres programmes distribués sur le même support, soient *un logiciel open-source.*

La licence doit être une technologie neutre

Aucune disposition de la licence ne peut être fondée sur une technologie ou un style de l'interface.

ANNEXE B : DIFFÉRENTS PROCESSUS DES PHASES DE REVUE DANS LE PROJET ECLIPSE

Le processus de développement de logiciels dans Eclipse a les caractéristiques d'être transparent et ouvert à tous les membres de la communauté Eclipse. Pour plus d'informations, il faut se référer au processus de développement dans Eclipse [7]. Le tableau B.1 présente les différentes phases de revues dans Eclipse.

Phases de revues	Descriptions, entrées, sorties	Étapes du processus de revues
1. Revue de création	**Entrée** : proposition de projet. **Sortie** : rapport de la revue	
2. Revue de graduation		1- Le EMO planifie la revue (2 fois par mois, généralement le mercredi) en envoyant un email à emo@eclipse.org. Les commentaires sont regroupés 4 à la fois 2- Au moins à une semaine à l'avance de la revue, l'auteur (du projet, ou le chef de la proposition de projet) envoie le *DocuWare* [55] à emo@eclipse.org en vue d'être posté 3- EMO annonce dès lors les revues via : http://www.eclipse.org/projects/whatsnew.php, ou http://www.eclipse.org/projects/reviews-rss.php, ou par la liste de diffusion de développeurs. Si un appel à conférence est requis, l'EMO le planifie, et informe les participants de la revue. L'appel dure 15 – 20 minutes. 4- Après un vote favorable du conseil et de l'approbation de la revue de graduation par l'EMO : 1. l'EMO envoie un courriel à la liste d'envoi des développeurs du projet avec une approbation 2. Le projet peut se retirer de l'incubation, avec l'étiquetage et les logos de leurs pages Web
3. Revue de version	C'est pour la validation d'une version du produit logiciel	Les étapes 1-3 sont le mêmes que celles du processus de la revue de graduation. Après un vote favorable du conseil et de l'approbation de la revue de graduation par

		Sorties : une version stable du produit, rapport de la revue/documentation du produit	l'EMO : 1. L'EMO envoie un courriel sur la liste de développement du projet confirmant des résultats concluants 2. Puis, lorsque les fichiers sont prêts pour la version finale, le projet peut les afficher sur le serveur de téléchargement et de laisser le puissant système de réplication distribuer les morceaux à travers l'univers connu
3.	**Revue de promotion**	Cette revue est faite dans le but de déterminer si le projet ayant été soumi, a démontré les caractéristiques d'un projet de haut-niveau	Processus non mentionné
4.	**Revue de continuité**	Elle a pour but d'aboutir à un projet viable, ou un crédit d'Eclipse	Processus non mentionné
5.	**Revue de terminaison**	C'est une dernière discussion sur une proposition de projet avant son retrait ou son archivage	Pour des informations, consulter [75]
6.	**Revue de migration**	Son but est de vérifier qu'il n'y a aucun obstacle juridique à la propriété intellectuelle du transfert du code d'un projet à un autre projet. Elle agit comme	Pour des informations, consulter [75]

		une élection (si nécessaire) pour les *Committers* qui sont ajoutés au nouveau projet	
7.	**Revue de restructuration**	Elle a pour but de vérifier qu'il n'existe aucun obstacle juridique de propriété intellectuelle de la restructuration proposée, et de donner la chance à la communauté sur la restructuration	Processus non mentionné
8.	**Revue de combinaison**	C'est la combinaison de plusieurs revues. Le plus souvent la combinaison rencontrée est : migration+version, migration+graduation, graduation+version	Processus non mentionné

Tableau B.1 Différentes phases de revue dans le projet Eclipse

ANNEXE C : ÉTAT DE L'ART DES MÉTHODOLOGIES DE DÉVELOPPEMENT

1 INTRODUCTION

Le cycle de développement de logiciels est la période partant de la définition des besoins du logiciel jusqu'à la fin ou arrêt de l'exploitation du logiciel [116]. Elle ne se résume donc pas juste à la phase de développement ou codage du logiciel. Chaque cycle de développement de logiciels contient les phases ou processus[36] suivants :

- **Expression des besoins :** Elle consiste à la description informelle des besoins exprimés par l'utilisateur et permet de définir le **cahier des charges**.
- **Spécification :** Elle est une concrétisation des besoins des utilisateurs. Cette phase consiste notamment en la description formelle du problème par l'équipe de développement.
- **Analyse, architecture et conception :** À ce niveau, l'on procède à la recherche de solutions tenant compte de l'architecture technique. Cela permet de définir le **comment**. Ce qui signifie de définie comment le logiciel sera réalisé. Habituellement, la conception est divisée en deux phases : la **conception globale** et la **conception détaillée**. La conception détaillée représente la conception du système global, composant par composant; tandis

[36] Il est à noter dans certains cas que, quelques uns de ces processus sont regroupés. Cependant, les différents processus cités sont toujours présents dans n'importe quel cycle de développement de logiciels. De plus, ils se déroulent toujours dans l'ordre cité.

que la conception globale donne une vision globale de ce que doit faire le système.

- **Développement ou implémentation:** Consiste principalement à la production du **code** du logiciel, tout en se référant à la phase d'analyse, d'architecture et de conception; mais sans avoir besoin de la remettre en question [116]. Dans cette phase, afin de s'assurer de la bonne marche du système, les développeurs effectuent des tests unitaires.
- **Tests et intégration:** Lors de la phase de tests, le client valide les fonctionnalités du système. L'intégration quant à elle consiste à regrouper tous les composants logiciels et de s'assurer que le tout ensemble, fonctionne parfaitement.
- **Déploiement/maintenance :** À ce niveau, les livrables ont été validés et sont déployés ou livrés afin que le client y ait accès. La maintenance corrective du produit logiciel ou la maintenance évolutive du produit logiciel peuvent être retrouvées à ce niveau.

Dans certains cas, lors du développement d'un système, quelques uns de ces processus de développement peuvent être regroupés. Cependant, les différents processus cités ci-dessus sont toujours présents dans n'importe quel cycle de développement de logiciels, et se déroulent toujours dans l'ordre cité. Par la suite, nous présenterons les différents cycles de vie de logiciels connus.

2 CYCLES DE VIE DE LOGICIELS

Il existe deux grandes familles de cycle de vie de logiciels : **les cycles séquentiels** et **les cycles itératifs**. Nous ne nous attarderons dans cette section que sur les cycles de vie séquentiels, car ils représentent les méthodes de développement dites **classiques**.

2.1 LES CYCLES SÉQUENTIELS

Les méthodes séquentielles de développement, dites **méthodes classiques** sont des méthodes grandement centrées sur des livrables ou documentation, fournis entre chaque étape du cycle de développement. Ceci permet de s'assurer de la réalisation d'un processus et ainsi permettre un meilleur suivi du projet. Cette façon de fonctionner dans les cycles séquentiels, rend ces modèles très peu populaires de nos jours, car le développement est très procédural et bureautique. Les méthodes dites séquentielles sont pour la plupart utilisées dans un développement fermé, en occurrence le développement de logiciels en entreprise.

Ces méthodes de développement nécessitent obligatoirement un contrat signé avant de débuter l'implémentation du logiciel, incluant les coûts et les délais du projet. Elles ne sont pas très adaptables aux besoins des clients, qui sont sans cesse changeant compte tenu de l'évolution de leur stratégie, et qui proviennent du fait que les clients ont du mal à exprimer leurs besoins sous une forme exploitable par l'équipe de développement [116].

2.1.1 Modèle en cascade

Il est l'un des premiers modèles de développement ayant vu le jour. Il a été formalisé par W. Royce en 1970. Issu de la construction de bâtiment dans lesquelles les fondations sont construites avant le reste de la maison, les activités de ce modèle sont exécutées de manière successive, les unes après les autres. Chaque phase ou activité, s'achève par la production d'un certain nombre de documents ou de logiciels. Ces documents ou logiciels sont nécessaires pour la tâche de vérification et de validation (tâche de V&V) qui permet de déterminer l'approbation du passage à la phase

suivante, ou la correction de l'étape précédente. Le modèle en cascade nécessite que les besoins soient clairs et précis. Il est adapté aux projets de petite taille, avec peu d'incertitude [116].

Il est représenté par la figure C.1 :

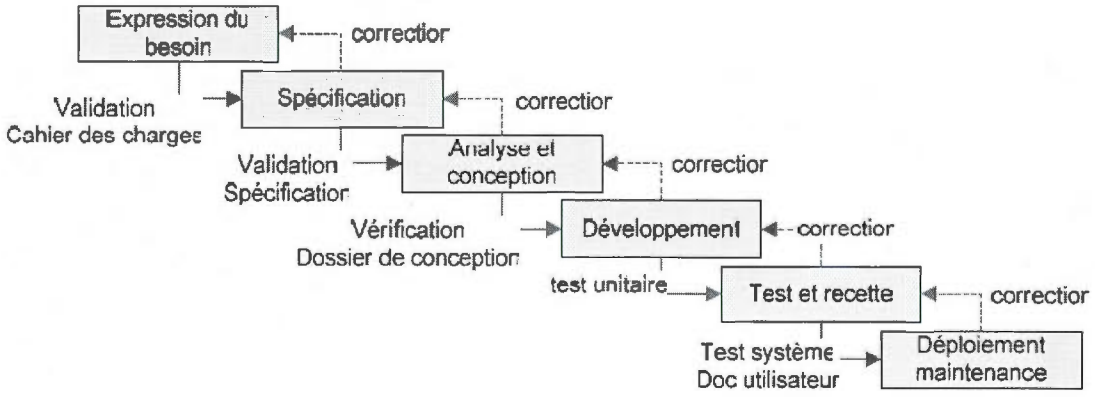

Figure C.1 Modèle en cascade

Thiessoz [116] a eu à identifier les avantages et les inconvénients de ce modèle :

Avantages :

- Modèle simple;
- Tests entre chaque phase;
- Gestion des phases en temps et ressources;
- Définition des tâches et des livrables.

Inconvénients :

- Effet tunnel (identification tardive des problèmes, preuve tardive du bon fonctionnement);
- Difficulté de cerner les besoins dès le début, refus de tout changement;
- Frustration de l'attente de la première version (lenteur);

- Les projets informatiques sont rarement séquentiels.

2.1.2 Modèle en V

Le modèle en V (voir figure C.2) est un modèle plus récent que le modèle en cascade, proposé en 1984 par Mc Dermid et Ripkin [116]. Il représente une amélioration du modèle en cascade par l'association à chaque phase de développement d'une phase de test. Le modèle en V confronte les différents niveaux de tests avec les phases de projets de même niveau, permettant ainsi à chaque étape, de définir non seulement les fonctions, mais également les critères de validation [37]. Il est adapté aux projets de taille moyenne [37].

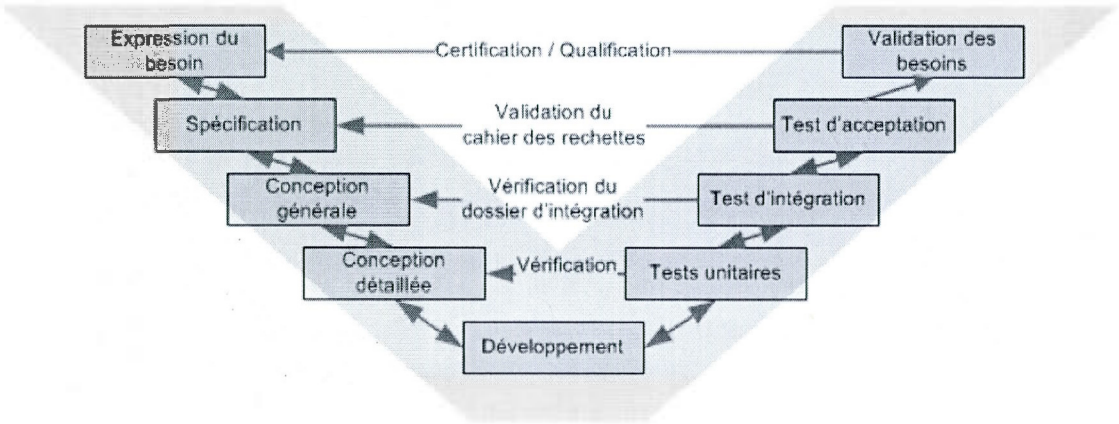

Figure C.2 Modèle en V

Thiessoz [116] a eu à identifier les avantages et les inconvénients de ce modèle :

Avantages :

- Réactivité accrue vis-à-vis du modèle en cascade (évite d'énoncer une propriété non vérifiable objectivement une fois le logiciel réalisé);
- Modèle industriel classique éprouvé;

- Validation systématique de chaque étape avec retour en arrière possible;
- La mise en place de la vérification lors de la conception oblige à une réflexion supplémentaire très utile.

En plus de ces avantages, nous notons les avantages suivants [37]:

- Identifier et anticiper très tôt les éventuelles évolutions des besoins;
- Permet de vérifier de la maturité des utilisateurs.

Inconvénients :

- Pas de prototypage (code/validation tardive) entrainant un risque d'effet tunnel;
- Prise en compte des modifications du cahier des charges difficile;
- Obligation de définir la totalité des besoins au départ;
- Projet rarement séquentiel.

2.1.3 Modèle par incréments

Il a été conçu afin d'adresser les limites du modèle en V. Le principe soutenant le modèle par incréments est donc de découper l'expression des besoins en sous-parties, communément appelées incréments. Thiessoz, dans son rapport mentionne que chaque incrément dans ce modèle, est réalisé successivement ou en se chevauchant selon un modèle en cascade ou en V [116].

Dans le modèle en cascade ou en V, il est implicite qu'après une décomposition en composants, ceux-ci sont développés indépendamment les uns des autres, en parallèle ou de manière séquentielle selon les ressources disponibles [53]. Dans ce modèle, seul un seul sous ensemble est développé à la fois [53].

Le modèle par incréments est représenté par la figure suivante :

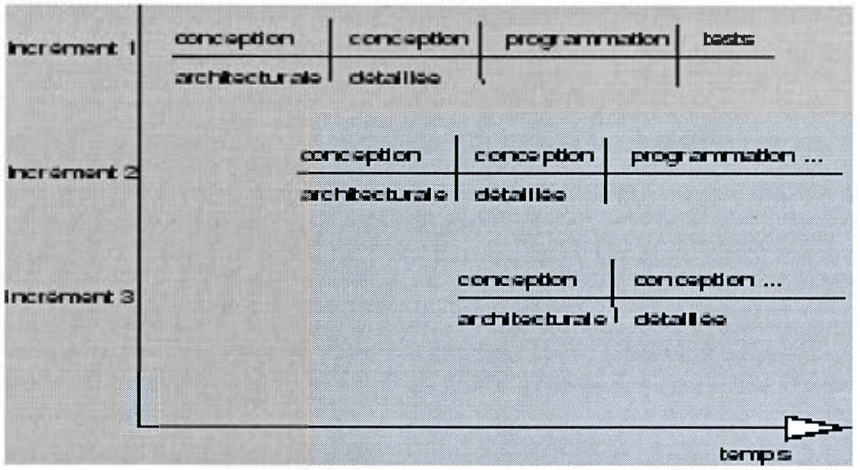

Figure C.3 Modèle par incréments tiré de [53]

Thiessoz [116] a eu à identifier les avantages et les inconvénients de ce modèle :

Avantages :

- Diminution de la durée d'un cycle et donc de l'effet tunnel;
- Diminution de la complexité (vision unique d'un incrément);
- Les intégrations sont progressives.

Inconvénients :

- Difficulté de l'intégration;
- Difficulté de la conception globale;
- Inadapté aux besoins d'évolution en cours de projet, problème en cas de remise en cause du noyau.

2.2 LES CYCLES ITÉRATIFS

Ce sont des cycles de vie plus souples que les cycles séquentiels. Ces cycles de développement sont dits « **méthodes agiles** » (en anglais *Agile Modeling*) car ils

visent à réduire le cycle de vie du logiciel. Et donc, ils permettent d'accélérer le développement du logiciel, en développant une version minimale, puis en intégrant les fonctionnalités par un processus itératif basé à la fois sur une écoute du client et sur des tests tout au long du cycle de développement du logiciel. Ces méthodes de développement sont de ce fait pragmatiques car elles visent la réelle satisfaction du client et non pas celle d'un contrat établi préalablement, comme tel est le cas pour les méthodes séquentielles [116].

Parmi les cycles itératifs, nous distinguons entre autre :

- **Le modèle en spiral** : Proposé par Boehm en 1988, ce cycle reprend les différentes étapes du modèle en V et se focalise sur l'analyse des risques. Ce modèle est basé sur des itérations. Lors de chaque nouvelle itération, les fonctionnalités du logiciel ayant déjà été implémentées lors de la précédente phase ou itération, sont ajustées ou raffinées.
- **Le modèle itératif** : Il a pour intérêt de se focaliser sur les fonctionnalités essentielles du logiciel et de le raffiner à chaque « tour de boucle » ou « itérations ». Le principe de ce modèle est d'appliquer un cycle de type « roue de Deming (voir la définition à [20]) » pour chaque nouveau besoin [116]. Ce cycle est effectué de manière itérative. L'idée ici, est de pouvoir livrer le plus tôt possible une version fonctionnelle du système au client, afin qu'il puisse la tester. De ce fait, les itérations doivent être les plus courtes possibles.

La figure C.4, présente quelques cycles de développement de logiciels.

Type	Cascade	Semi-itératif	Itératif
Descriptifs	Approche totalement cascade	Approche en cascade dans la première partie. Puis approche itérative-incrémentale pour la gestion des besoins	Totalement itérative-incrémentale avec une pré-phase d'exploration
Exemples	Merise, SDM/S	RAD, RUP	XP, DSDM, Crystal

Figure C.4 Quelques exemples de cycles de développement adapté de [46, 116]

ANNEXE D : ISO/IEC TR 29110 – 1

SOFTWARE ENGINEERING – LIFECYCLE PROFILES FOR VERY SMALL ENTITIES (VSES) – PART1: OVERVIEW

NB : Cette partie de la norme a été copiée telle qu'elle. Elle peut être téléchargée gratuitement sur le lien suivant : http://standards.iso.org/ittf/licence.html

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example), it may decide to publish a Technical Report. A Technical Report is entirely informative in nature and shall be subject to review every five years in the same manner as an International Standard.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 29110-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.
ISO/IEC 29110 consists of the following parts, under the general title *Software engineering — Lifecycle profiles for Very Small Entities (VSEs)*:

- *Part 1: Overview* [Technical Report]
- *Part 2: Framework and taxonomy*
- *Part 3: Assessment guide* [Technical Report]
- *Part 4-1: Profile specifications: Generic profile group*
- *Part 5-1-2: Management and engineering guide: Generic profile group: Basic profile* [Technical Report]

Entry profile, intermediate profile and advanced profile will form the subjects of future Parts 5-1-1, 5-1-3 and 5-1-4, respectively.

Parts 4 and 5 can be developed to accommodate new profile specifications and management and engineering guides as follows:

- *Part 4-m: Profile specifications: Profile group aaaaa*
- *Part 5-m-n: Management and engineering guide: Profile group aaaaa: Profile bbbbb* [Technical Report]

Introduction

The software industry recognizes the value of Very Small Entities (VSEs) in contributing valuable products and services. For the purpose of ISO/IEC 29110, a Very Small Entity (VSE) is an entity (enterprise, organization, department or project) having up to 25 people. VSEs also develop and/or maintain software that is used in larger systems; therefore, recognition of VSEs as suppliers of high quality software is often required.

According to the Organization for Economic Co-operation and Development (OECD) SME and Entrepreneurship Outlook report (2005), 'SMEs constitute the dominant form of business organization in all countries world-wide, accounting for over 95 % and up to 99 % of the business population depending on country'. The challenge facing OECD governments is to provide a business environment that supports the competitiveness of this large heterogeneous business population and that promotes a vibrant entrepreneurial culture.

From studies and surveys conducted, it is clear that the majority of International Standards do not address the needs of VSEs. Conformance with these standards is difficult, if not impossible. Subsequently VSEs have no, or very limited, ways to be recognized as entities that produce quality software in their domain. Therefore, VSEs are often cut off from some economic activities.

It has been found that VSEs find it difficult to relate International Standards to their business needs and to justify the application of the standards to their business practices. Most VSEs can neither afford the resources, in terms of number of employees, budget and time, nor do they see a net benefit in establishing software lifecycle processes. To rectify some of these difficulties, a set of guides has been developed according to a set of VSE characteristics. The guides are based on subsets

of appropriate standards elements, referred to as VSE Profiles. The purpose of a VSE Profile is to define a subset of International Standards relevant to the VSE context, for example, processes and outcomes of ISO/IEC 12207 and products of ISO/IEC 15289.

ISO/IEC 29110, targeted by audience, has been developed to improve product and/or service quality, and process performance. See Table 1. ISO/IEC 29110 is not intended to preclude the use of different lifecycles such as: waterfall, iterative, incremental, evolutionary or agile.

Table 1 — ISO/IEC 29110 target audience

ISO/IEC 29110	**Title**	**Target audience**
Part 1	Overview	VSEs, assessors, standards producers, tool vendors, and methodology vendors
Part 2	Framework and taxonomy	Standards producers, tool vendors and methodology vendors. Not intended for VSEs.
Part 3	Assessment guide	Assessors and VSEs
Part 4	Profile specifications	Standards producers, tool vendors and methodology vendors. Not intended for VSEs.
Part 5	Management and engineering guide	VSEs

If a new profile is needed, ISO/IEC 29110-4 and ISO/IEC TR 29110-5 can be developed without impacting existing documents and they become ISO/IEC 29110-4-*m* and ISO/IEC 29110-5-*m*-*n*, respectively, through the ISO/IEC process.

This part of ISO/IEC 29110 defines the business terms common to the ISO/IEC 29110 series. It introduces processes, lifecycle and standardization concepts, and the ISO/IEC 29110 series. It also introduces the characteristics and requirements of a VSE, and clarifies the rationale for VSE-specific profiles, documents, standards and guides.

ISO/IEC 29110-2 introduces the concepts for software engineering standardized profiles for VSEs, and defines the terms common to the ISO/IEC 29110 series. It establishes the logic behind the definition and application of standardized profiles. It specifies the elements common to all standardized profiles (structure, conformance, assessment) and introduces the taxonomy (catalogue) of ISO/IEC 29110 profiles.

ISO/IEC TR 29110-3 defines the process assessment guidelines and compliance requirements needed to meet the purpose of the defined VSE Profiles. ISO/IEC TR 29110-3 also contains information that can be useful to developers of assessment methods and assessment tools. ISO/IEC TR 29110-3 is addressed to people who have direct relation with the assessment process, e.g. the assessor and the sponsor of the assessment, who need guidance on ensuring that the requirements for performing an assessment have been met.

ISO/IEC 29110-4-*m* provides the specification for all the profiles in one profile group that are based on subsets of appropriate standards elements. VSE Profiles apply and are targeted to authors/providers of guides and authors/providers of tools and other support material.

ISO/IEC TR 29110-5-*m*-*n* provides an implementation management and engineering guide for the VSE Profile described in ISO/IEC 29110-4-*m*.

Figure 1 describes the ISO/IEC 29110 series and positions the parts within the framework of reference. Overviews and guides are published as Technical Reports (TR), and profiles are published as International Standards (IS).

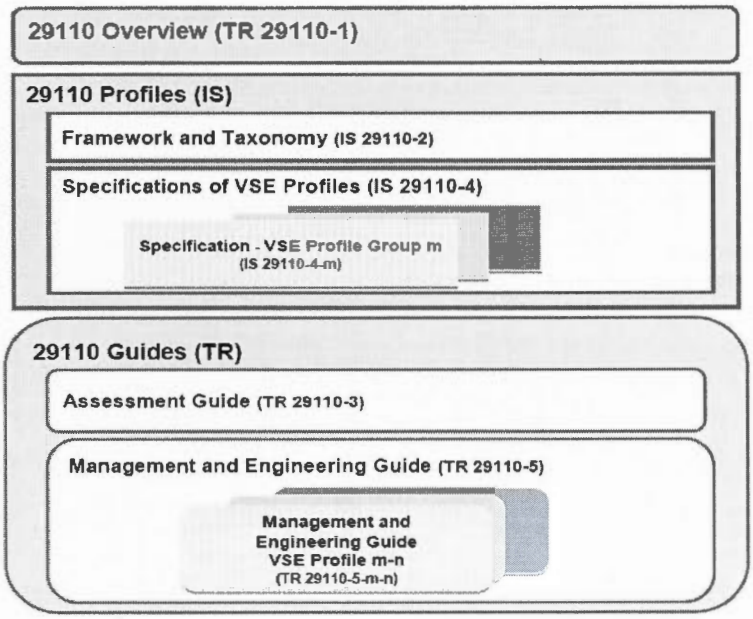

Figure 1 — ISO/IEC 29110 series

Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 1: **Overview**

1. Scope

1.1. Fields of application

This part of ISO/IEC 29110 introduces the major concepts required to understand and use the ISO/IEC 29110 series. It introduces the characteristics and requirements of a Very Small Entity (VSE), and clarifies the rationale for VSE-specific profiles, documents, standards and guides.

It also introduces process, lifecycle and standardization concepts and defines the business terms common to the ISO/IEC 29110 series.

This part of ISO/IEC 29110 is applicable to VSEs. The lifecycle processes described in ISO/IEC 29110 are not intended to preclude or discourage their use by an entity that is larger than a VSE.

1.2. Target audience

This part of ISO/IEC 29110 is targeted both at the general audience wishing to understand the ISO/IEC 29110 series and more specifically at users of the ISO/IEC 29110 series. It is intended that it be read first when initially exploring VSE Profile documents. While there is no specific prerequisite to read this part of ISO/IEC 29110, it will be helpful to the user in understanding the other parts.

The lifecycle processes defined in ISO/IEC 29110 can be used by a VSE when acquiring and using, as well as when creating and supplying, software. They can be

applied at any level in a software system's structure and at any stage in the lifecycle. They are not intended to preclude or discourage the use of additional processes that a VSE finds useful.

2. Terms and definitions

For the purposes of this document, the following terms and definitions apply.

2.1. Activity

Set of cohesive tasks of a process

[ISO/IEC 12207:2008]

2.2. Assessment indicator

Sources of objective evidence used to support the assessors' judgment in rating process attributes.

EXAMPLE Work products, practice, or resource.

[ISO/IEC 15504-1]

2.3. Assessor

Individual who participates in the rating of process attributes

[ISO/IEC 15504-1]

2.4. Baseline

Specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures

[ISO/IEC 12207:2008]

2.5. Base standard

Approved International Standard or Telecommunication Standardization Sector of the International Telecommunications Union (ITU-T) Recommendation

[ISO/IEC TR 10000-1]

2.6. Competent assessor

Assessor who has demonstrated the competencies to conduct an assessment and to monitor and verify the conformance of a process assessment

[ISO/IEC 15504-1]

2.7. Customer

Organization or person that receives a product or service

NOTE A customer can be internal or external to the organization.

[ISO/IEC 12207:2008]

2.8. Deployment package

Set of artefacts developed to facilitate the implementation of a set of practices of the selected framework.

2.9. Generic profile group

Profile group applicable to VSEs that do not develop critical software products and have typical situational factors

2.10. Guide

Document published by ISO or IEC giving rules, orientation, advice or recommendations relating to international standardization

[ISO/IEC Directives, Part 2]

2.11. International standard

Standard that is adopted by an international standardizing/standards organization and made available to the public

[ISO/IEC Directives, Part 2]

2.12. Standardized profile

Internationally agreed-to, harmonized standard which describes one or more profiles

NOTE Adapted from the definition of "International Standardized Profile" in ISO/IEC TR 10000-1.

2.13. Lifecycle

Evolution of a system, product, service, project or other human-made entity from conception through retirement

[ISO/IEC 12207:2008]

2.14. Process

Set of interrelated or interacting activities which transforms inputs into outputs

[ISO 9000]

2.15. Process assessment

Disciplined evaluation of an organizational unit's processes against a Process Assessment Model

[ISO/IEC 15504-1]

2.16. Process assessment model

Model suitable for the purpose of assessing process capability, based on one or more Process Reference Models

[ISO/IEC 15504-1]

2.17. Process capability

Characterization of the ability of a process to meet current or projected business goals

[ISO/IEC 15504-1]

2.18. Process capability level

Point on the six-point ordinal scale (of process capability) that represents the capability of the process, each level building on the capability of the level below

[ISO/IEC 15504-1]

2.19. Process improvement

Actions taken to change an organization's processes so that they more effectively and/or efficiently meet the organization's business goals

[ISO/IEC 15504-1]

2.20. Process outcome

Observable result of a process

[ISO/IEC 15504-1]

2.21. Process profile

Set of process attribute ratings for an assessed process

[ISO/IEC 15504-1]

2.22. Process reference model

Model comprising definitions of processes in a lifecycle described in terms of process purpose and outcomes, together with an architecture describing the relationships between the processes

[ISO/IEC 15504-1]

2.23. Profile

Set of one or more base standards and/or profiles and, where applicable, the identification of chosen classes, conforming subsets, options and parameters of those base standards or standardized profiles necessary to accomplish a particular function

[ISO/IEC TR 10000-1]

2.24. Profile group

Collection of profiles which are related by composition of processes (i.e. activities, tasks) or by capability level, or both

2.25. Project

Endeavour with defined start and finish dates undertaken to create a product or service in accordance with specified resources and requirements

[ISO/IEC 12207:2008]

2.26. Record

Document stating results achieved or providing evidence of activities performed

[ISO/IEC 20000-1]

2.27. Report

Information item that describes the results of activities such as investigations, assessments, and tests

[ISO/IEC 15289:2006]

2.28. Repository

Collection of all software-related artefacts belonging to a system or the location/format in which such a collection is stored

[ISO/IEC/IEEE 24765]

2.29. Resource

Asset that is utilized or consumed during the execution of a process

[ISO/IEC 12207:2008]

2.30. Review

Process or meeting during which a software product is presented to project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval

[IEEE Std 1028]

2.31. Software

Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system

[IEEE Std 829]

2.32. Software component

Software system or element, such as module, unit, data, or document

[IEEE Std 1061]

2.33. Standard

Document, established by consensus and approved by a recognized body, that provides, for common and repeated use, rules, guidelines or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context

NOTE Standards should be based on the consolidated results of science, technology and experience, and aimed at the promotion of optimum community benefits.

[ISO/IEC Directives, Part 2]

2.34. Taxonomy

Classification scheme for referencing profiles or sets of profiles unambiguously

[ISO/IEC TR 10000-1]

2.35. Technical Report

Document published by ISO or IEC containing collected data of a different kind from that normally published as an International Standard or Technical Specification

[ISO/IEC Directives, Part 2]

NOTE Such data may include, for example, data obtained from a survey carried out among the national bodies, data on work in other international organizations or data on the "state of the art" in relation to standards of national bodies on a particular subject.

2.36. Traceability record

Work product that

- identifies requirements to be traced,
- identifies a mapping of requirement to lifecycle work products,
- provides the linkage of requirements to work product decomposition (i.e. requirement, design, code, test, deliverables, etc.),
- provides forward and backwards mapping of requirements to associated work products throughout all phases of the lifecycle

NOTE 1 This may be included as a function of another defined work product (example: A CASE tool for design decomposition may have a mapping ability as part of its features).

NOTE 2 Definition taken from ISO/IEC 15504-5:2006, Annex B.

2.37. Task

Requirement, recommendation or permissible action intended to contribute to the achievement of one or more outcomes of a process

[ISO/IEC 12207:2008]

2.38. User

Individual or group that benefits from a system during its utilization

[ISO/IEC 12207:2008]

2.39. Validation

Confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled

[ISO 9000]

NOTE Validation in a lifecycle context is the set of activities ensuring and gaining confidence that a system is able to accomplish its intended use, goals and objectives.

2.40. Verification

Confirmation, through the provision of objective evidence, that specified requirements have been fulfilled

[ISO 9000]

NOTE Verification in a lifecycle context is a set of activities that compares a product of the lifecycle against the required characteristics for that product. This may include, but is not limited to, specified requirements, design description and the system itself.

2.41. Very small entity

VSE

Entity (enterprise, organization, department or project) having up to 25 people

2.42. Work product

Artefact associated with the execution of a process

[ISO/IEC 15504-1]

3. Conventions and abbreviated terms

3.1. Naming, diagramming and definition conventions

None.

3.2. Abbreviations

VSE Very Small Entity
VSEs Very Small Entities

4. VSE Characteristics and VSE Potential Benefits

4.1. General

A VSE is considered to be an entity that engages in software implementation activities regardless of its legal form. An entity can be an organization (registered or non-registered), a group within an organization, or a project within an organization. Organization can mean an independent partnership or linked organization having up to 25 people that is engaged in a software implementation project. Annex A provides more basic information.

4.2. VSE Characteristics

VSEs are subject to a number of characteristics, needs and desirable competencies that affect the contents, the nature and the extent of their activities. The VSE Profiles address VSEs which are described through the following characteristics, needs and desirable competencies, classified in four categories: Finance and Resources, Customer Interface, Internal Business Processes and Learning and Growth.

In some cases, a VSE is expected to perform limited missions in the entire software development lifecycle under the directions of either another company or consortium fulfilling contract or agreement requirements. These missions may be a part of the software implementation project according to the statement of work. The VSE is chosen by its own competencies or by a bid for the project. Amplification of these characteristics is provided in ISO/IEC 29110-4 set of documents.

4.3. VSE Potential Benefits

From the VSE perspective, some benefits considered for using the ISO/IEC 29110 series include good internal software management processes, greater customer confidence and satisfaction, greater software product quality, increased sponsorship

for process improvement and decreased development risk. These benefits might also help with increased competitiveness and market share.

5. Lifecycle Process Concepts

5.1. Introduction

This clause provides lifecycle process concepts that are considered in the ISO/IEC 29110 series and are supportive of the potential coordinated use of ISO/IEC 12207 and ISO/IEC 15289. It will assist users in their management of information items as products of the system or software lifecycle.

5.2. Lifecycle Models and Stages

This sub-clause helps to establish a common framework for software lifecycle processes and in planning, producing, and evaluating the results of the lifecycle processes.

5.1.12. Life Cycle Models and Stages

The life of a system or a software product can be modelled by a life cycle model consisting of stages. Models may be used to represent the entire life from concept to disposal or to represent the portion of the life corresponding to the current project. The life cycle model is comprised of a sequence of stages that may overlap and/or iterate, as appropriate for the project's scope, magnitude, complexity, changing needs and opportunities. Each stage is described with a statement of purpose and outcomes. The life cycle processes and activities are selected and employed in a stage to fulfil the purpose and outcomes of that stage. Different organizations may undertake different stages in the life cycle. However, each stage is conducted by the organization responsible for that stage with due consideration of the available information on life cycle plans and decisions made in preceding stages. Similarly, the organization responsible for that stage records the decisions made and records the assumptions regarding subsequent stages in the life cycle.

This International Standard does not require the use of any particular life cycle model. However, it does require that each project define a suitable life cycle model, preferably one that has been defined by the organization for use on a variety of projects. Application of a life cycle model provides the means to establish the time-dependent sequence necessary for project management.

[ISO/IEC 12207]

5.3. Lifecycle Product Types

This sub-clause helps to clarify that information items are essential to preserving what transpired when using system lifecycle processes and be identified as deliverable documents. The result of a process shall be documented or may imply the need for a document (or information item) and often do not specify the contents.

The use of generic types simplifies the application of consistent structure, content, and formats for similar information items (records and documents), to support usability. ISO/IEC 29110 defines the lifecycle data of ISO/IEC 12207:2008 and ISO/IEC 15288:2008 by relating tasks and activities to the generic information item types given in Table 2.

Table 2 — Lifecycle product types

Type	**Purpose**	**Sample of recommended output information types**
Record	Characterize the data an organizational entity retains.	Configuration record Problem record
Description	Represent a planned or actual function, design, or item	High-level software design description
Plan	Define when, how, and by whom specific activities are to be performed.	Project management plan
Procedure	Define in detail when and how to perform certain activities or tasks, including tools needed.	Problem resolution procedure
Report	Describe the results of activities such as investigations, assessments, and tests.	Problem report Validation report
Request	Record information needed to solicit a response.	Change request
Specification	Specify a required function, performance or process (such as,	Software requirements specification

	requirements specification, standard, policy).	

NOTE Adapted from ISO/IEC 15289:2006.

6. Process Improvement and Assessment Concepts

6.1. Process Improvement Concepts

The process improvement concept is to encourage a VSE's project teams to implement systematic approaches which allow for the repetition and realism in estimating and implementing a project. Periodic assessments and communication (internal and external) of the project progress will ensure customer satisfaction.

Process improvement concepts characterize all actions undertaken to improve an organization's processes to both increase their efficiency and meet the organization's business goals. Process improvement activities are addressed in ISO/IEC 15504 and ISO/IEC 29110-3.

Ideally, process improvement is driven by business goals such as increasing productivity, customer satisfaction or increasing market share. Several approaches start with business goals identification, followed by identification of potential problems preventing the realization of these business goals. From this diagnosis, corrections are identified and implemented.

It could be easy for a VSE to over commit to a specific customer project based on their limited resources. Periodic assessments and communication (internal and external) of the project progress will help ensure customer satisfaction.

6.2. Assessment Concepts

Assessment concept refers to the determination of the extent to which the organization's processes contribute to the achievement of its business goals and to help the organization focus on the need for process improvement. For example, the assessment can be either formal or informal, use an outside evaluator or an internal evaluator, use a standard checklist or personnel interviews, etc.

Traditional assessments can be very expensive. Pre-assessment activities such as preparation of documents to prove work has been done correctly, allotment of personnel time for review and internal management of the assessment can be a drain on resources in a VSE.

A simpler method for VSEs might be a combination of self-assessment and spot checks to verify actual practices followed, without having an independent assessor on site for a full assessment. However, a case can be made that the VSE needs to do an assessment to satisfy customer concerns regardless of whether a formal certification to a standard is sought.

7. Standardization Concepts

7.1. Introduction

Recognizing the limitations of VSEs resources, the need for minimum processes and practices are supported in the scope of ISO/IEC 29110. This will allow the VSE to be flexible and achieve its business goals without compromising software engineering processes.

NOTE Rationale for defining each profile is in ISO/IEC 29110-2.

7.2. Standard

Software engineering standards are focusing on both processes and products aspects. They contain formal requirements developed and used to prescribe consistent approaches to develop software. Software Engineering Standards have several objectives:

- to provide a common framework and vocabulary for software project practitioners;
- to provide a framework for two party agreements;
- to improve and evaluate software competence;
- to facilitate software process or product evaluation.

Standards contain normative and, in some cases, informative parts. The normative part of standards is used as requirements for conformance evaluation. The informative part of standards contains information that either complements or facilitates the understanding of or the use of the normative part.

The need for process improvement should be a business issue, motivated by profitability. In large organizations, large quantities of data are tracked in sophisticated ways, including application of Lean and six sigma tools. In VSE's process improvement can be handled, more informally.

7.3. Technical Reports

Technical Reports are documents published by ISO/IEC JTC 1 containing collected data of a different kind from that normally published as an International Standard or Technical Specification. It contains information that helps in the understanding and use of the normative part of a standard.

In the scope of ISO/IEC 29110, Technical Reports are used to present the guidelines information about implementing the profile and assessing its implementation in a VSE.

Capability assessments were created to address the low confidence that acquirers of large systems had in software developers, i.e., governments purchasing large weapons systems. VSE's often have a domain specific product that is needed by a larger enterprise. Oftentimes these products are available as COTS products, proven by years of in-service history. Such history can be used in lieu of costly capability assessments.

7.4. Profile

A profile is a set of one or more base standards and/or standardized profiles, and, where applicable, the identification of chosen classes, conforming subsets, options and parameters of those base standards, or standardized profiles necessary to accomplish a particular function.

7.5. Profile Group

A profile group (PG) is a collection of profiles which are related either by composition of processes (i.e. activities, tasks), or by capability level, or both. Amplification of these characteristics is provided in ISO/IEC 29110-2.

7.6. Generic Profile Group

The generic profile group is applicable to VSEs that do not develop critical software products. The generic profile group does not imply any specific application domain. Amplification of these characteristics is provided in ISO/IEC 29110-2.

7.7. Guides

Guides provide practical information to facilitate the implementation and the assessment of the implementation of the defined profiles. In accordance with JTC 1, guides are published as Technical Reports.

7.8. Use of Profiles

Profiles are designed to be used by a VSE to implement specific functionality through the use of guides published as Technical Reports. At a minimum, each Profile of the ISO/IEC 29110 series shall be linked to an Assessment Guide and one or more implementation guides.

Additional materials, such as a deployment package, a set of artefacts developed to facilitate the implementation of a set of practices of the selected framework, is available to further facilitate the implementation of Profiles by a VSE.

NOTE Annex A of the Engineering and Management Guides (ISO/IEC 29110 TR 5-1-*n*) gives additional information about deployment packages.

Any document that has been developed from a profile shall reference the standardized profile they were derived from.

7.9. Conformance to Profiles

Conformance to profiles may be complete, when all the required elements of the profile are satisfied or partially satisfied when a selected subset is completed.

Conformance with a profile implies compliance with the selected components of the base standards.

For methodology related products and tools, conformance means that the proposed method or tools implements the required elements of the profile.

For the implementation of the required elements within a VSE, conformance means that actual performance of the processes can be evaluated through an assessment process.

These concepts provide the context and standardization details for the format and content of the product, as required to support the principles and classification schema selected for them. Conformance to a VSE Profile should be evaluated through an assessment defined in the assessment guide referenced by the VSE Profile.

Conformance to a VSE Profile is the way that VSEs show and document their use and understanding of International Standards. By conforming to the guidance of International Standards, the VSE shows that the content of their produced documents are compliant with the support the concepts required by these standards.

8. ISO/IEC 29110 series

8.1. Introduction

The ISO/IEC 29110 series is comprised of multiple documents with different purposes and audiences. Documents are grouped in three categories overview, profiles and guides. The overview document is the introductory document for the set of other documents. The profile documents are the technical specifications for the packaging of the various profile elements. The guide documents are the user oriented documents. Figure 1 identifies the major categories and the existing and planned documents.

8.2. Overview

The overview introduces all the major concepts required to understand and use the ISO/IEC 29110 series. It introduces the characteristics and requirements of a VSE, and clarifies the rationale for VSE-specific profiles, documents, standards and guides. It also introduces process, lifecycle and standardization concepts, and the ISO/IEC 29110 series. It is targeted both at a general audience interested in these documents, and more specifically at users of these documents. The overview document is identified as ISO/IEC 29110-1.

8.3. VSE Profiles

VSE Profiles are defined to formally package references to other documents and/or parts of other documents in order to adapt them to a VSE needs and characteristics. Preparing VSE Profiles is an ISO/IEC JTC 1 defined process. It involves producing two types of documents, the Framework and Taxonomy plus Profile Specifications

8.3.1. Framework and Taxonomy

The framework and taxonomy document establishes the logic behind the definition and application of process profiles. It specifies the elements common to all process

profiles (structure, conformance, assessment) and introduces the taxonomy (catalogue) of ISO/IEC 29110 profiles. The framework and taxonomy document is applicable to all profiles and is identified as ISO/IEC 29110-2.

8.3.2. Profile Specifications

There is a profile specification document for each profile. The profile specification document purpose is to provide the formal composition of a profile, provide normative links to the normative subset of standards (e.g. ISO/IEC 12207:2008) used in the profile, and to provide informative links (references) to "input" documents. There is one profile specification document for each profile group, which is identified as ISO/IEC 29110-4-*m*, where *m* is the number assigned to the profile group.

8.4. Guides

Guides contain implementation guidelines (domain specific) on how to perform the processes to achieve the maturity levels (e.g. recommended activities, measures, techniques, templates, models, methods, etc.). Guides are developed for the process implementation and assessment based on the domain's issues, business practices and risks. Guides are targeted at VSEs, and should be VSE accessible, both in terms of style and cost.

8.4.1. Assessment Guide

The assessment guide describes the process to follow in performing an assessment to determine the process capabilities. This is used when an organization wants an assessment performed in order to obtain a process capability profile of the implemented processes and/or an organizational process maturity level. It is also applicable if a customer asks for a third-party assessment evaluation. This could also

be used to obtain a capability level profile of the implemented process by the software implementation and maintenance provider and is also suitable for a self-assessment. The assessment guide is applicable to all profiles and is identified as ISO/IEC 29110-3.

8.4.2. Management and Engineering Guides

The management and engineering guides provide guidance for its implementation and use of a profile. They are targeted at VSE management and technical staff and VSE-related organizations such as technology transfer centres, government industry ministries, national standards, consortiums and associations, academic use for training, and authors of derived products (software, courseware, acquirer and suppliers). There is one management and engineering guide document for each profile within each profile group, identified as ISO/IEC 29110-5-*m*-*n*, where *m* is the number assigned to the profile group and *n* the number assigned to the profile. This number matches the number assigned to the profile specification.

Annex A

(informative)

Basic reference works

A.1. Rationale

The software industry recognizes the value of very small entities in contributing valuable products and services to economy. As software quality increasingly becomes a subject of concern, and process approaches are maturing and gaining the confidence of companies, the use of ISO/IEC standards is spreading in organizations of all sizes. However, these standards were not written for development organizations with less than 25 people, and are consequently difficult to apply in such small settings.

This proposal aims to address those difficulties by developing profiles and by providing guidance for conformance with ISO/IEC software engineering standards. This framework will attempt to ease the use of ISO/IEC 12207 processes and ISO 9001, and reduce the conformance obligations by providing VSE Profiles. The framework will develop guidance for each process profile and provide a roadmap for conformance with ISO/IEC 12207 and ISO 9001.

A.2. Market Study

A market survey of VSEs was conducted to ask questions about their utilization of ISO/IEC standards. The purpose of the survey was to collect data to identify problems and potential solutions to help VSEs apply ISO/IEC standards and become more competitive. The survey underlined that there are three main reasons preventing VSEs from using ISO/IEC standards. The first is a lack of resources (28%); the

second is that standards are not required (24%); and the third derives from the nature of the standards themselves: 15% of the respondents consider that the standards are difficult and bureaucratic, and do not provide adequate guidance for use in a small business environment. However, for a large majority (74%) of VSEs, it is very important to be recognized or certified against a standard. ISO certification is requested by 40% of them. However, VSEs are expressing the need for assistance in order to adopt and implement standards. Over 62% would like more guidance with examples, and 55% are asking for lightweight and easy-to-understand standards complete with templates.

A.3. Current Standards

Since the 1980s the software process engineering knowledge area has seen the emergence of various software process standards whose main goal was to provide a way to assess and improve software processes in organizations. Among those standards are ISO/IEC 12207 and ISO/IEC 15504 from the International Organization for Standardization/ International Electrotechnical Commission. Research institutes like the Software Engineering Institute (SEI), in close partnership with industry, developed a de facto framework, the Software Capability Maturity Model Integration (CMMI).

Each of these standards has been developed to answer problems within a particular context, at its own time. For example, in the 1980s, almost all software projects of the US Department of Defence (DoD) were over time and budget. Therefore, the DoD asked the SEI to provide leadership in assisting software organizations to develop and continuously improve their capability to identify, adopt, and use sound management and technical practices.

As is the case with the CMMI, all those standards fit better organizations that are similar to the ones for which they were developed and tested. Their use in other organizations like a VSE is often more complicated. In practice, an organization starting a process improvement approach has to select one of the available models and furthermore has to use it as adequately as possible. But this organization has its own particular context that should be taken into account in this choice and in the improvement approach. In particular, the selection and the adaptation tasks become laborious in the case of VSEs.

Bibliography

[1] ISO 9000, *Quality management systems — Fundamentals and vocabulary*

[2] ISO 9001, Quality management systems — Requirements

[3] ISO/IEC TR 10000-1, Information technology — Framework and taxonomy of International Standardized Profiles — Part 1: General principles and documentation framework

[4] ISO/IEC 12207:2008, Systems and software engineering — Software life cycle processes

[5] ISO/IEC 15288:2008, Systems and software engineering — Systems life cycle processes

[6] ISO/IEC 15289:2006, Systems and software engineering — Content of systems and software life cycle process information products (Documentation)

[7] ISO/IEC 15504, Information technology — Process assessment

[8] ISO/IEC 20000-1, Information technology — Service management — Part 1: Service management system requirements

[9] ISO/IEC/IEEE 24765, Systems and software engineering — Vocabulary

[10] ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards, Sixth edition, 2011

[11] IEEE Std 829, IEEE Standard for Software Test Documentation

[12] IEEE Std 1028, IEEE Standard for Software Reviews and Audits

[13] IEEE Std 1061, IEEE Standard for a Software Quality Metrics Methodology

[14] CONRADI, R., DYBA, T., SJOBERG, D., ULSUND, T., *Software Process Improvement. Results and Experience from the Field*, Springer, 2006

[15] Organization for Economic Co-Operation and Development (OECD), *SME and Entrepreneurship Outlook*, 2005 Edition

[16] RICHARDSON, I., GRESSE VON WANGENHEIM, Ch., Guest Editors' Introduction: Why are Small Software Organizations Different? in *IEEE Software*, vol. **24**, no. 1, pp. 18-22, Jan/Feb, 2007

[17] OKTABA, H and PIATTINI, M., Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies, Idea Group Inc, Hershey, PA, 2008

[18] LAPORTE, C.Y., RENAULT, A., ALEXANDRE, S., UTHAYANAKA, T., *The Application of ISO/IEC JTC 1/SC 7 Software Engineering Standards in Very Small Enterprises*, ISO Focus, International Organization for Standardization, September 2006, pp. 36-38

ANNEXE E : ISO/IEC TR 29110 – 3

SOFTWARE ENGINEERING – LIFECYCLE PROFILES FOR VERY SMALL ENTITIES (VSES) – PART3: ASSESSMENT GUIDE

NB : Cette partie de la norme a été copiée telle qu'elle. Elle peut être téléchargée gratuitement sur le lien suivant : http://standards.iso.org/ittf/licence.html

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example), it may decide to publish a Technical Report. A Technical Report is entirely informative in nature and shall be subject to review every five years in the same manner as an International Standard.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 29110-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.
ISO/IEC 29110 consists of the following parts, under the general title *Software engineering — Lifecycle profiles for Very Small Entities (VSEs)*:

- *Part 1: Overview* [Technical Report]

- *Part 2: Framework and taxonomy*

- *Part 3: Assessment guide* [Technical Report]

- *Part 4-1: Profile specifications: Generic profile group*

- *Part 5-1-2: Management and engineering guide: Generic profile group: Basic profile* [Technical Report]

Entry profile, intermediate profile and advanced profile will form the subjects of future Parts 5-1-1, 5-1-3 and 5-1-4, respectively.

Parts 4 and 5 can be developed to accommodate new profile specifications and management and engineering guides as follows:

- *Part 4-m: Profile specifications: Profile group aaaaa*

- *Part 5-m-n: Management and engineering guide: Profile group aaaaa: Profile bbbbb* [Technical Report]

Introduction

The software industry recognizes the value of Very Small Entities (VSEs) in contributing valuable products and services. For the purpose of ISO/IEC 29110, a Very Small Entity (VSE) is an entity (enterprise, organization, department or project) having up to 25 people. VSEs also develop and/or maintain software that is used in larger systems; therefore, recognition of VSEs as suppliers of high quality software is often required.

According to the Organization for Economic Co-operation and Development (OECD) SME and Entrepreneurship Outlook report (2005) 'SMEs constitute the dominant form of business organization in all countries world-wide, accounting for over 95 % and up to 99 % of the business population depending on country'. The challenge facing OECD governments is to provide a business environment that supports the competitiveness of this large heterogeneous business population and that promotes a vibrant entrepreneurial culture.

From studies and surveys conducted, it is clear that the majority of International Standards do not address the needs of VSEs. Conformance with these standards is difficult, if not impossible. Subsequently VSEs have no, or very limited, ways to be recognized as entities that produce quality software in their domain. Therefore, VSEs are often cut off from some economic activities.

It has been found that VSEs find it difficult to relate International Standards to their business needs and to justify the application of the standards to their business practices. Most VSEs can neither afford the resources, in terms of number of employees, budget and time, nor do they see a net benefit in establishing software lifecycle processes. To rectify some of these difficulties, a set of guides has been developed according to a set of VSE characteristics. The guides are based on subsets

of appropriate standards elements, referred to as VSE Profiles. The purpose of a VSE Profile is to define a subset of International Standards relevant to the VSE context, for example, processes and outcomes of ISO/IEC 12207 and products of ISO/IEC 15289.

ISO/IEC 29110, targeted by audience, has been developed to improve product and/or service quality, and process performance. See Table 1. ISO/IEC 29110 is not intended to preclude the use of different lifecycles such as: waterfall, iterative, incremental, evolutionary or agile.

Table 1 — ISO/IEC 29110 target audience

ISO/IEC 29110	**Title**	**Target audience**
Part 1	Overview	VSEs, assessors, standards producers, tool vendors and methodology vendors
Part 2	Framework and taxonomy	Standards producers, tool vendors and methodology vendors. Not intended for VSEs.
Part 3	Assessment guide	Assessors and VSEs
Part 4	Profile specifications	Standards producers, tool vendors and methodology vendors. Not intended for VSEs.
Part 5	Management and engineering guide	VSEs

If a new profile is needed, ISO/IEC 29110-4 and ISO/IEC TR 29110-5 can be developed without impacting existing documents and they become ISO/IEC 29110-4-*m* and ISO/IEC 29110-5-*m*-*n*, respectively, through the ISO/IEC process.

ISO/IEC TR 29110-1 defines the business terms common to the ISO/IEC 29110 series. It introduces processes, lifecycle and standardization concepts, and the ISO/IEC 29110 series. It also introduces the characteristics and requirements of a VSE, and clarifies the rationale for VSE-specific profiles, documents, standards and guides.

ISO/IEC 29110-2 introduces the concepts for software engineering standardized profiles for VSEs, and defines the terms common to the ISO/IEC 29110 series. It establishes the logic behind the definition and application of standardized profiles. It specifies the elements common to all standardized profiles (structure, conformance, assessment) and introduces the taxonomy (catalogue) of ISO/IEC 29110 profiles.

This part of ISO/IEC 29110 defines the process assessment guidelines and compliance requirements needed to meet the purpose of the defined VSE Profiles. This part of ISO/IEC 29110 also contains information that can be useful to developers of assessment methods and assessment tools. This part of ISO/IEC 29110 is addressed to people who have direct relation with the assessment process, e.g. the assessor and the sponsor of the assessment, who need guidance on ensuring that the requirements for performing an assessment have been met.

ISO/IEC 29110-4-*m* provides the specification for all the profiles in one profile group that are based on subsets of appropriate standards elements. VSE Profiles apply and are targeted to authors/providers of guides and authors/providers of tools and other support material.

ISO/IEC TR 29110-5-*m*-*n* provides an implementation management and engineering guide for the VSE Profile described in ISO/IEC 29110-4-*m*.

Figure 1 describes the ISO/IEC 29110 series and positions the parts within the framework of reference. Overviews and guides are published as Technical Reports (TR), and profiles are published as International Standards (IS).

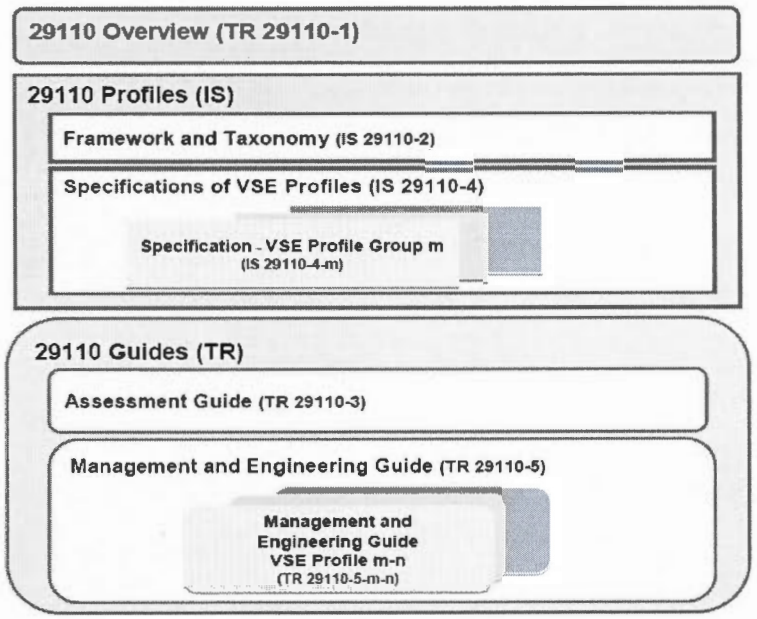

Figure 1 — ISO/IEC 29110 series

Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 3: Assessment guide

1. Scope

1.1. Fields of application

This part of ISO/IEC 29110 defines the process assessment guidelines and conformance requirements needed to meet the purpose of defined VSE Profiles. It is applicable to all VSE Profiles and is compatible with ISO/IEC 15504-2.

The possible uses of this part of ISO/IEC 29110 are as follows.

a) Assessment to evaluate the process capabilities. This is when an organization wants an assessment execution in order to obtain a process profile of the implemented processes.

b) Provider's capability assessment. This is when a customer asks for a third party to conduct an assessment in order to obtain a process profile of the implemented process by the software development and maintenance provider. The customer chooses the processes to be assessed depending on the services to be contracted.

1.2. Target audience

The target audience of this part of ISO/IEC 29110 is primarily those who perform process assessments for VSEs. This part of ISO/IEC 29110 also contains information that can be useful to developers of assessment methods and assessment tools.

This part of ISO/IEC 29110 is addressed to people who have a direct relation with the assessment process based on the VSE Profiles, e.g. the assessor and the sponsor of the assessment, who need guidance on ensuring that the requirements for performing an assessment have been met.

It is intended that ISO/IEC TR 29110-1 and ISO/IEC 29110-2 be read first when initially exploring VSE Profile documents.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC TR 29110-1, *Software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 1: Overview*

3. Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC TR 29110-1 apply.

4. Conventions and abbreviated terms

4.1. Naming, diagramming and definition conventions

None.

4.2. Abbreviations

VSE Very Small Entity
VSEs Very Small Entities

5. Assessment framework

These guidelines apply to VSE Process Assessments. The assessment, as defined in this part of ISO/IEC 29110 and in the Standardized Profiles for VSE, has two purposes:

To evaluate the process capability based on a two dimensional assessment model containing a process dimension and a capability dimension. The process dimension refers to the processes defined in each VSE Profile which are provided by an external Process Reference Model. The capability dimension consists of a Measurement Framework comprising six Process Capability Levels and their associated Process Attributes.

To evaluate whether an organization achieves the targeted VSE Profile based on the evaluated capabilities for the processes.

For any official recognition, the assessments should be carried out following an assessment process satisfying the requirements of ISO/IEC 15504-2 and described in Clause 6 of this part of ISO/IEC 29110. For self-assessments emphasizing identification of process improvements, other approaches can be applied (additional information can be found in ISO/IEC 29110-5).

According to ISO/IEC 15504-2, a process assessment is a disciplined evaluation of an organizational unit's processes against a Process Assessment Model. In this context, the Process Assessment Model consists of a subset of process purposes and outcomes of a Process Reference Model, and the process attributes that are defined in ISO/IEC 15504-2:2003. A Process Reference Model is, for instance, ISO/IEC 12207:2008 and the applicable subset is defined in a Specification of a VSE Profile, for instance, ISO/IEC 29110-4-1. The applied Process Assessment Model needs to be conformant to ISO/IEC 15504-2. The result of a process assessment is represented as a set of process attribute ratings, i.e. a process profile. Figure 2 illustrates the relevant documents and data for a process applicable to VSE process assessment.

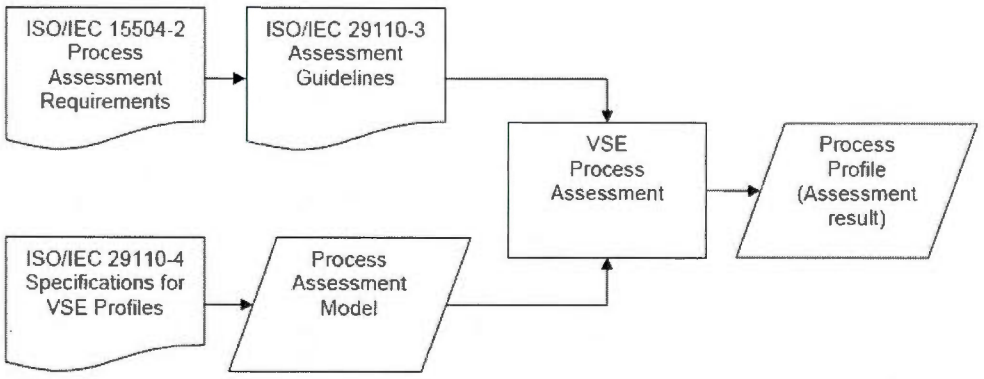

Figure 2 — Elements of VSE process assessment

ISO/IEC 15504-2 sets out the minimum requirements for performing an assessment that ensure consistency and repeatability of the ratings. The requirements help to ensure that the assessment output is self-consistent and provides evidence to substantiate the ratings and to verify conformance with the requirements. Detailed guidance on performing an assessment is available in ISO/IEC 15504-3:2003.

Self-assessments are typically performed to identify process improvement opportunities or to check current status of the organization's performance. Self-assessments in VSEs are outside the scope of this part of ISO/IEC 29110.

6. VSE process assessment

6.1. Performing an assessment

6.1.1. Introduction

In performing an assessment based on ISO/IEC 29110, the requirements expressed in ISO/IEC 15504-2 are intended to be satisfied in full. The assessors should be familiar with the intent of these requirements and the guidance provided in ISO/IEC 15504-3. This clause of ISO/IEC 29110-3 provides additional guidance related specifically to the process assessment in VSEs.

An assessment is conducted according to a documented process that is capable of meeting the assessment purpose. The key elements of a documented assessment process are closely tied to the requirements for performing an assessment, defined in Clause 4 of ISO/IEC 15504-2:2003. The documented assessment process is the set of instructions for conducting the assessment. A documented assessment process addresses the following aspects of the conduct of an assessment:

- defining the inputs to an assessment such as purpose, scope, constraints and the identity of the conformant Process Assessment Model to be used;
- defining key roles and responsibilities;
- providing guidance for planning, data collection, data validation, process attributes rating and reporting of assessment results;
- recording of assessment outputs.

6.1.2. Assessment inputs

Assessment inputs as specified in ISO/IEC 15504-2:2003, 4.4 are to be defined. In conducting assessments of VSEs based on ISO/IEC 29110, the following issues are of importance:

- The process scope of the assessment [ISO/IEC 15504-2, 4.4.2 (c) (1, 2)] is determined by the VSE Profile specified for the assessment.
- The organizational scope of the assessment [ISO/IEC 15504-2, 4.4.2 (c) (3)] will typically be the entire VSE; however, where the VSE deploys a small number of clearly distinct projects or functions, the scope may be limited to a single project or function.
- In defining the assessment context [ISO/IEC 15504-2, 4.4.2 (c) (4)], the assessment plan should take into account the VSE business and engineering context and be affordable to a VSE.
- In defining the assessment constraints [ISO/IEC 15504-2, 4.4.2 (e)], the specific nature of the VSE should be explored to establish constraints on availability of resources or data that might affect the reliability of the assessment.

6.1.3. Roles and responsibilities

Typically, the assessment team for VSE assessment process consists of at least one competent assessor or a competent assessor with other assessors. The assessors should be familiar with the VSE characteristics.

6.1.4. The assessment process

The activities to be performed will be determined by the chosen documented assessment process tailored as necessary. The documented process for the assessment of a VSE should address all of the required activities defined in ISO/IEC 15504-2:2003, 4.2.2.

Specific concerns of relevance to assessment of VSEs include the following:

Planning

Typically, the schedule for assessment of a VSE will need to take account of the availability of key resources. The level of resources required for the assessment should be determined according to the resources available to the VSE.

Data collection

The strategy for data collection should take account of the nature of the work performed within the VSE, and of the nature of the items of objective evidence that will typically be available. Often assessments in VSEs rely heavily on testimony from performers of the processes; however, to the best extent possible, the assessors should endeavour to obtain other supporting objective evidence drawn from the VSE work products.

Data validation

The key issue in data validation in assessment of a VSE is ensuring that the data collected is representative of the normal operations of the enterprise.

Process attribute rating

In conducting process attribute rating, the assessors should focus on the extent to which the evidence obtained addresses the processes and process attributes being rated. The requirement for traceability between the rating and the evidence employed [ISO/IEC 15504-2:2003, 4.2.2 d) 4)] is relevant here.

Reporting

The assessors should ensure that the report to the sponsor of the assessment covers the full scope of the VSE Profile employed in the assessment.

6.2. Use of the assessment results

The assessment results may be used to:

a) Evaluate the process capabilities of an organization,

b) Determine the improvement opportunities, in order to enhance the organization's ability to meet its business goals by improving efficiency and quality of its software products and services. The findings can be used as a base to perform the improvement plan,

c) Benchmark the process capabilities with other organizations in the market,

d) Select a provider based on the provider's capability assessment.

6.3. Achievement of a VSE Profile

This section provides guidance on how to determine whether an organization has achieved a VSE Profile. The determination is based on the evaluated capabilities for the processes in each VSE Profile. ISO/IEC 29110-4 Clause 2.2 defines the conformance requirements.

The requirements for the achievement of VSE Profiles can be derived from the respective parts of ISO/IEC 29110-4 and ISO/IEC 29110-5. At minimum, all

mandatory elements of the VSE Profile, as defined in ISO/IEC 29110-4 , need to be considered in the assessment.

To achieve, for example, the Basic VSE Profile the assessed processes need to achieve capability level one as defined in ISO/IEC 15504-2. This means that the implemented process achieves its process purpose and its defined outcomes. The applicable process purposes are documented in ISO/IEC 29110-5-1-2:2011:

- Project Management process purpose in clause 6.1 PM Purpose, and

- Software Implementation process purpose in clause 7.1 SI Purpose.

The related outcomes of the Process Reference Model ISO/IEC 12207 are documented in ISO/IEC 29110-5-1-2 under the process specific Objectives. A detailed mapping of the Basic VSE Profile process elements to ISO/IEC 12207 and other Base Documents is provided for in Clause 8 of ISO/IEC 29110-4-1:2011.

6.4. Application of process assessment models

Use of ISO/IEC 15504-2 compliant Process Assessment Model (PAM) ensures that the assessment results are comparable, reliable and repeatable. The assessor should confirm that the applied PAM is suitable for assessing the process capability in the context of VSEs.

The applied Process Assessment Model should have a set of indicators that address the process purpose and outcomes, and demonstrate the achievement of the required capability level.

ISO/IEC 29110-4 Specifications for VSE Profiles document a detailed mapping of process elements between part 5 of ISO/IEC 29110 and the Process Reference Models.

ISO/IEC 15504-5 is an Exemplar Process Assessment Model based on ISO/IEC 12207 Process Reference Model and it can be applied to assess the capability of the VSE Profiles. ISO/IEC 15504-5 has a complete mapping of the assessment indicators to the ISO/IEC 12207 process outcomes. A VSE specific PAM can be derived by selecting those assessment indicators relevant to the corresponding process outcomes defined in ISO/IEC 29110-4.

Bibliography

[1] ISO/IEC 12207:2008, *Systems and software engineering — Software life cycle processes*

[2] ISO/IEC 15504-1:2004, Information technology — Process assessment — Part 1: Concepts and vocabulary

[3] ISO/IEC 15504-2:2003, Information technology — Process assessment — Part 2: Performing an assessment

[4] ISO/IEC 15504-3:2004, Information technology — Process assessment — Part 3: Guidance on performing an assessment

[5] Organization for Economic Co-Operation and Development (OECD), *SME and Entrepreneurship Outlook*, 2005 Edition.

BIBLIOGRAPHIE

[1] [ANONYME], « A Framework for Evaluating OpenSource Software », *BBR*, http://www.openbrr.org/, consulté le 12 Avril 2011.

[2] [ANONYME], « About Us », *Mozilla*, http://www.mozilla.org/about/, consulté le 24 Mars 2011.

[3] [ANONYME], « Apache Infrastructure », *The Apache Software Foundation*, http://www.apache.org/dev/infrastructure.html, consulté le 20 Janvier 2011.

[4] [ANONYME], « Artefact », *Wikipedia, l'encyclopédie libre*, n° 76674800, 15 Mars 2012, http://fr.wikipedia.org/wiki/Artefact, consulté le 08 Mai 2012.

[5] [ANONYME], « Automation », *Mozilla*, http://quality.mozilla.org/teams/automation/, consulté le 20 Janvier 2011.

[6] [ANONYME], « Contributor License Agreement», *The Apache Software Foundation*, http://www.apache.org/licenses/#clas, consulté le 13 Juin 2011.

[7] [ANONYME], « Eclipse Development Process », *The Eclipse Foundation*, http://www.eclipse.org/projects/dev_process/development_process.php#3_1_Requirements_and_Guidelines, consulté le 25 Juillet 2010.

[8] [ANONYME], « FreeBSD », *Wikipedia, l'encyclopédie libre*, n° 78394886, 2011, http://fr.wikipedia.org/wiki/FreeBSD, consulté le 10 Février 2011.

[9] [ANONYME], « How the ASF works », *The Apache Software Foundation*, http://www.apache.org/foundation/how-it-works.html#structure, consulté le 23 Mars 2011.

[10] [ANONYME], « Informations concernant le système de traitement de bogues pour les responsables de paquets et trieurs de bogues », *Debian*, http://www.debian.org/Bugs/Developer, consulté le 28 Janvier 2011.

[11] [ANONYME], « ISO 12207 », *Wikipedia, l'encyclopédie libre*, http://en.wikipedia.org/wiki/ISO_12207, consulté le 20 Janvier 2011.

[12] [ANONYME], « Liste de logiciels libres », *Wikiepedia, l'encyclopédie libre*, n° 77954558, 2012, http://fr.wikipedia.org/wiki/Liste_de_logiciels_libres, consulté le 01 Mai 2012.

[13] [ANONYME], « Logiciel libre : une étude prévoit plus de 40% de croissance en 2010 en Europe », *AFP*, 11 mars 2010, http://www.google.com/hostednews/afp/article/ALeqM5gxbYQwjRUpJKpXZ1hSuKWeZwYpNg, consulté le 10 Août 2010.

[14] [ANONYME], « Mozilla Organisations », *Mozilla*, http://www.mozilla.org/about/organizations.html, consulté le 2 Février 2011.

[15] [ANONYME], « Mozilla Roles and Leadership », *Mozilla*, http://www.mozilla.org/about/roles.html, consulté le 2 Février 2011.

[16] [ANONYME], « Noyau Linux », *Wikipedia, l'encyclopédie libre*, n° 77969531, 2011, http://fr.wikipedia.org/wiki/Noyau_Linux, consulté le 01 Septembre 2011.

[17] [ANONYME], « philosophie », *Larousse*, http://www.larousse.fr/dictionnaires/francais/philosophie, consulté le 17 Juin 2011.

[18] [ANONYME], « Qu'est ce qu'un logiciel libre », *Misfu*, http://www.misfu.com/definition-logiciel-libre.html, consulté le 14 Août 2010.

[19] [ANONYME], « Roles », *The Apache Foundation*, http://www.apache.org/foundation/how-it-works.html#roles, consulté le 23 Mars 2011.

[20] [ANONYME], « Roue de Deming », *Wikipedia, l'encyclopédie libre*, n° 77827880, 2011, http://fr.wikipedia.org/wiki/Roue_de_Deming, consulté le 19 Juin 2011.

[21] [ANONYME], « The Linux Kernel Archives», *The Linux Kernel Organisation Inc.*, http://www.kernel.org/, consulté le 05 Juillet 2011.

[22] [ANONYME], « The Mozilla Manifesto », *Mozilla*, http://www.mozilla.org/about/manifesto, consulté le 12 Avril 2011.

[23] [ANONYME], « The Open Source Definition », *Open Source Initiative*, http://www.opensource.org/docs/osd, consulté le 10 Août 2010.

[24] [ANONYME], « Valeur (personnelle et culturelle) », *Wikipedia, l'encyclopédie libre*, n° 77733115, 2011, http://fr.wikipedia.org/wiki/Valeur_%28personnelle_et_culturelle%29, consulté le 12 Mai 2011.

[25] [ANONYME], « Welcome to ASF Bugzilla », *The Apache Software Foundation*, https://issues.apache.org/bugzilla/, consulté le 20 Janvier 2011.

[26] Aberdour, Mark, « Achieving Quality in Open Source Software », *IEEE Computer Society, IEEE Software*, vol. 24, n° 1, 2007, p. 58-64, ISSN 0740-7459, USA.

[27] Abran, Alain, Janes W. Moore, Pierre Bourque et Robert Dupuis, « Guide to the Software Engineering Body of Knowledge », *IEEE Computer Society*, vol. 1, n° 1, 2004, www.swebok.org .

[28] Ahmed, Mohamed F., et Swapna S. Gokhale, « Linux Bugs: Live Cycle and Resolution Analysis », *IEEE computer society*, n° 150-6002, 2008, USA.

[29] Apache HTTP Server Project, *Apache Software Foundation*, http://httpd.apache.org/, consulté le 05 Juillet 2011.

[30] Apache, « The Apache Software Foundation », www.apache.org, consulté le 04 Août 2010.

[31] Asundi, Jai, et Rajiv Jayant, « Patch Review Processes in Open Source Software Development Communities: A comparative Case Study », *Proceedings of the 40th Annual Hawaii International Conference on System Sciences, IEEE Computer Society*, 2007.

[32] Baruch, Siach, « Linux Kernel Participation How to», 2011, http://www.slideshare.net/benavrhm/linux-kernel-participation-howto, consulté le 16 Mars 2012.

[33] Bauval, Anne, et Sisyph, « Compatibilité », *Wikipedia, l'encyclopédie libre*, n° 74776853, 26 Janvier 2012, http://fr.wikipedia.org/wiki/Compatibilit%C3%A9, consulté le 01 Mai 2012.

[34] Behlendorf, B., « Open source as a business strategy », cité par Chris Diboma, Sam Ockman et Mark Stone, *Open Sources: Voices from the Open Source Revolution*, O'Reilly and Associates, Cambridge, 1999.

[35] Behlendorf, B., Scott Bradner, Jim Hamerly, Marshal K. McKusick, Tim O'Reilly, Tom Paquin, Bruce Perens, Eric S. Raymond, Richard Stallman, Michael Tiemann, Linus Torvalds, Paul Vixie, Larry Wall, Bob Young, Chris Dibona, Sam Ockman et Mark Stone, *Tribune Libre : Ténors de l'Informatique Libre*, O'Reilly & Associates, Inc., Éd 2, 978-2-35209-033-5, Novembre 2006, Éd. 1, ISBN 2-84177-084-2, Juin 1999, Adaptation française du livre intitulé *Open Sources: Voices from the Open Source Revolution*, 1999, USA, version gratuite en ligne : http://www.velic.com/publications/tribunelibre/index.html, consulté le 02 Avril 2011.

[36] Bollinger, Terry, Russel Nelson, Karsten M. Self et Stephen J. Turnbull, « Open Source Methods: Peering through the Clutter », *IEEE Software*, vol. 16, n° 4, 1999, p. 8-11.

[37] Capirossi, Jerôme, « Cycle de vie en V », http://capirossi.org/info/prj_mgt/cycle_3.htm, consulté le 13 Juin 2011.

[38] Crowston, Kevin, Hala Annabi, James Howison et Chengetai Masango, « Effective Work Practices for Software Engineering: Free/Libre Open Source Software Development », *ACM, in proceeding of WISER, NFS Grants*, n° 03-41475 et 04-14468, 2004, USA.

[39] D'Elia Blanco, Marcelo, « Logiciel libre », Texte extrait du livre : *les Enjeux de Mots*, http://vecam.org/article707.html, Mai 2006, consulté le 10 Août 2010.

[40] Debian GNU/Linux, www.debian.org, consulté le 10 Octobre 2011.

[41] Dépôt de documents GNOME, http://live.gnome.org/Git, consulté le 20 Janvier 2011.

[42] Dibona, Chris, Sam Ockman, Mark Stone, Eric S. Raymond, Marshall K. McKusick, Scott Bradner, Richard Staman, Michael Tiemann, Paul Vixie, Linus Torvalds, Robert Young, Larry Wall, Brian Behlendorf, Bruce Perens, Tim O'Reilly, Jim Hamerly, Tom Paquin et Susan Walton, *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates, Inc., n° 1565925823, 1999, p. 280, Canada.

[43] Dictionnaire de l'académie française (8ème édition), « Définition de principe », *Mediadico*, http://www.mediadico.com/dictionnaire/definition/Principe/1, consulté le 10 Février 2011.

[44] Duenas, Juan C., Hugo A. Parada, Félix Cuadrado, Manuel Santillan et José L. Ruiz, « Apache and Eclipse: Comparing Open Source Project Incubators », *IEEE Computer Society*, IEEE Software, vol. 24, n° 6, 2007, p. 90-98.

[45] Dumais, Michel, « Avantages et contraintes du logiciel libre », *logiquelibre.com*, Avril 2007, http://www.logiquelibre.com/modules/documentation/item.php?itemid=3, consulté le 12 Juillet 2010.

[46] Duquesne, « Cycle de développement (logiciel) », *Wikipedia, l'encyclopédie libre*, n° 77536740, 2012, http://fr.wikipedia.org/wiki/Cycle_de_d%C3%A9veloppement, consulté le 08 Mai 2012.

[47] Eclipse, www.eclipse.org, consulté le 25 Juillet 2010.

[48] Elliot, Magaret, « A virtual Organisation Culture of a free Software Development Community », *3rd Workshop on Open Source Software Engineering*, 2003, USA.

[49] Erenkrantz, Justin R., « Release Management Within Open Source Projects », *3rd Workshop on Open Source Software Engineering*, 2003, USA.

[50] Fagan, M. E., « Design and code inspections to reduce errors in program development », *IBM Systems Journal*, vol. 15, n° 3, 1976, p. 182-211.

[51] Fielding, Roy T., « Shared Leadership in the Apache Project », *Communication of ACM*, vol. 42, n° 4, 1999, p. 42-43, USA.

[52] Fielding, Roy T., et G. Kaiser, « The Apache HTTP Server Project », *IEEE Internet Computing*, vol. 1, n° 4, 1997, p. 88-90.

[53] Fournier, Jean-Pierre, « Modèle par incréments », http://www.infeig.unige.ch/support/se/lect/gl/models/node15.html, consulté le 13 Juin 2011.

[54] Free Software Fondation, www.fsf.org, consulté le 14 Août 2010.

[55] Freeman-Benson, Bjorn, « Development Resources/FAQs/About Docuware », *The Eclipse Foundation*, http://wiki.eclipse.org/Development_Resources/FAQs/About_Docuware, consulté le 25 Juillet 2010.

[56] Gacek, C., et B. Arief, « The many meanings of Open Source ». *IEEE Software* 21, p. 34–40, 2004 cité par Flore Barcillini, Francoise Détienne et Jean-Marie Burkhardt, «User and developer mediation in an Open Source Software community: Boundary spanning through cross participation in online discussions », *International Journal of Human-Computer Studies*, vol. 66, n° 7, 2008, p. 558 – 570.

[57] GCC, http://gcc.gnu.org/, http://directory.fsf.org/project/gcc/, consulté le 05 Juillet 2011.

[58] Georg, von Krogh, Sebastian Spaeth et Karim R. Lakhani, « Community, joining, and specialization in open source software innovation: a case study », *Research Policy* in Open Source Software Development, vol. 32, n° 7, 2003, p. 1217–1241, http://dx.doi.org/10%252e1016/S0048%252d7333%252803%252900050%252d7.

[59] German, Daniel M., « The evolution of the GNOME Project », *Proceedings of the 2nd Workshop on Open Source Software Engineering*, 2002, Boston, MA.

[60] German, Daniel M., « The GNOME Project: A Case Study of Open Source Global Software Development », *Software Process Improvement and Practice*, vol. 8, n° 8, 2003, p. 201-215, Canada.

[61] GNOME, http://www.gnome.org/, http://directory.fsf.org/project/gnome/, consulté le 05 Juillet 2011.

[62] GNU, « Licences Commentées », http://www.gnu.org/licenses/license-list.fr.html, consulté le 12 Octobre 2010.

[63] Golden, Bernard, « Succeeding with Open Source », *Pearson Education Inc.*, n° 0-321-26853-9, 2005, Boston.

[64] Görlitz, Walter, « Comparison of issue-tracking systems », *Wikipedia, l'encyclopédie libre*, 2012, http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems, consulté le 08 Mai 2012.

[65] Guido Hertel, Sven Niedner et Stefanie Herrmann, «Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel », *Open Source Software Development*, vol. 32, n° 7, 2003, p. 1159–1177.

[66] Gwendolyn K.Lee et Robert E.Cole, « From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development », *Organization Science,* Vol. 14, n° 6, 2003, p. 633-649.

[67] Halloran, T. J., et W. L. Scherlis, « High quality and open source software practices », *Proceedings of the 2nd Workshop on Open Source Software Engineering*. Orlando, FL, USA: ICSE, 2002, USA.

[68] IEEE Std 1028-1997, « IEEE Standard for Software Reviews », *IEEE*, n° 1-55937-987-1, 1998, USA.

[69] IEEE Std1059-1993, « IEEE Guide for Software Verification and Validation Plans », *IEEE*, n° 0-7381-2379-X, 1994, USA.

[70] Ioannis, Samoladas, et Ioannis Stamelos, « Assessing Free/Open Source Software Quality », *Computer and Information Science*, Aristotle University of Informatics Greece nd, 2003, p. 1-36, USA, http://kb.cospa-project.org/retrieve/2768/samoladasstamelos.pdf , consulté le 12 Avril 2011.

[71] ISO/CEI 9126, « Technologie de l'information – Qualité des produits logiciels – Partie 1 : Modèle de qualité », 2001.

[72] ISO/IEC 24765:2010, « Systems and Software Engineering Vocabulary », *Organisation internationale de normalisation/Commission électrotechnique internationale*, Genève, 2010, en ligne : www.computer.org/sevocab.

[73] ISO/IEC PDTR 29110-5-1.3: 2009, « VSE Management and Engineering Guide for Basic Profile », nouvelle version: « ISO/IEC TR 29110-5-1-2:2011- Software Engineering - Lifecycle Profiles for Very Small Entities (VSEs) - Part 5-1-2: Management and engineering guide - Generic profile group: Basic profile », *Organisation internationale de normalisation/Commission électrotechnique international*, Genève, Suisse. Disponible gratuitement sur l'ISO: http://standards.iso.org/ittf/PubliclyAvailableStandards/c051153_ISO_IEC_TR_29110-5-1_2011.zip.

[74] Jaap, Boender, Roberto D. Cosmo, Jerôme Vouillon, Berke Durak et Fabio Mancinelli, « Improving the quality of GNU/Linux distributions », *IEEE Computer Society, COMPSAC*, n° 0730-3157, 2008, p. 1240 - 1246, Turku.

[75] Jacko, Anne, Pawel Piech, Bjorn Freeman-Benson et John Arthorne, « Development Resources/HOWTO/Review Information for Project Leads », *The Eclipse Foundation*, 2010, http://wiki.eclipse.org/Development_Resources/HOWTO/Review_Information_for_Project_Leads#Termination_.28Archive.29_Reviews, http://wiki.eclipse.org/Development_Resources/HOWTO/Review_Information_for_Project_Leads#Move_Reviews, consulté le 25 Juillet 2010.

[76] K Dictionaries Ltd., « Principe », *The free Dictionnary*, http://fr.thefreedictionary.com/principe, consulté le 10 Février 2011.

[77] Kempf, Jean-Baptiste, « la liste des meilleurs alternatives pour windows », *Logiciels Open source pour Windows ! et Faciles à Utiliser!*, http://oss.jbkempf.com/list.php, consulté le 01 Août 2011.

[78] Koch, Stefan, et Georg Schneider, « Effort, co-operation and co-ordination in an open source software project: GNOME », *Information System Journal*, vol. 12, n° 1, 2002, p. 27- 42.

[79] Koch, Stefan, et Georg Schneider, « Results from Software Engineering Research into Open Source Development Projects Using Public Data », *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft*, 22. Institut für Informationsverarbeitung und Informationswirtschaft, WU Vienna University of Economics and Business, Vienna, Austria, 2000, http://epub.wu.ac.at/494/, consulté le 10 Octobre 2011.

[80] Lakhani, Karim R., et R.G. Wolf, « Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects », *MIT Sloan Working Paper*, n° 4425-03, 2003, cité par Joseph Feller, Brian Fitzgerald, Scott A. Hissam et Karim A. Lakhani (eds.), « Making Sense of the Bazaar: Perspectives on Free and Open Source Software », *MIT Press*, 2005, p. 3-22, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=443040, consulté le 13 Août 1011.

[81] Laporte, Claude Y., et E. Palza Vargas, «The Development of International Standards to facilitate Process Improvements for Very Small Enterprises » Cité dans « Software Process Improvement and Management: Approaches and Tools for Practical Development », *IGI Global Publisher*, 2012, p. 34-61, USA. Disponible à: http://profs.etsmtl.ca/claporte/Publications/Publications/IGI_Chapter_SPI_in_VSEs.pdf.

[82] Le Bars, Christophe, « Logiciel libre et évolution », http://www.linux-france.org/article/these/libreevol/, version 199803, 1998.

[83] Lerner, Josh, et Jean Triole, « The Simple Economics of Open Source », *National Bureau of Economic Research: Cambridge*, n° 7600, 2000, MA, http://papers.nber.org/.

[84] Licences libres, http://www.gnu.org/licenses/, http://www.gnu.org/licenses/license-list.html#SoftwareLicenses, consulté le 12 Juillet 2010.

[85] Listes de liste de diffusion Eclipse : https://dev.eclipse.org/mailman/listinfo, consulté le 20 Janvier 2011.

[86] Listes de liste de diffusion Mozilla : https://lists.mozilla.org/listinfo, consulté le 20 Janvier 2011.

[87] Llias, Chris, « Understanding The Relationship And History Between Mozilla And Netscape », *Mozilla*, http://ilias.ca/MozillaNetscapeRelationship, consulté le 23 Mars 2011.

[88] Longchamp, Jacques, « Open Source Software Development Process Modeling », *SpringerLink, The Kluwer International Series in Software Engineering*, vol. 10, n° 0-387-24262-7_2, 2005, p. 29-64, France.

[89] MacCormack, A., R. Verganti et M. Iansiti, « Developing product on 'internet time': The anatomy of a flexible development process », *Management Science*, vol. 47, n° 1, 2001, p. 133–150, cité par Martin Michlmayr, « Quality Improvement in Volunteer Free and Open Source Software Projects : Exploring the Impact of Release Management », *Centre de la gestion de technologie, A dissertation submitted to the University of Cambridge for the Degree of Doctor of Philosophy*, 2007, Angleterre, http://www.cyrius.com/publications/michlmayr-phd.pdf, consulté le 20 Janvier 2011.

[90] McConnell, Steve, « Open Source Methodology: Ready for Prime Time? », *IEEE Software*, vol. 16, n° 4, 1999, p. 6-8.

[91] Michlmayr, Martin, « Quality Improvement in Volunteer Free and Open Source Software Projects : Exploring the Impact of Release Management », *Centre de la gestion de technologie, A dissertation submitted to the University of Cambridge for the Degree of Doctor of Philosophy*, 2007, Angleterre, http://www.cyrius.com/publications/michlmayr-phd.pdf, consulté le 20 Janvier 2011.

[92] Michlmayr, Martin, Francis Hunt et David Probert, « Quality Practices and Problems in Free Software Projects », *Proceedings of the First International Conference on Open Source Systems*, Juillet 2005, p. 24-28, Genève, Suisse.

[93] Michlmayr, Martin,« Quality Improvement in volunteers Free Software Projects: Exploring the Impact of Release Management », *Proceedings of the First International Conference on Open Source Systems*, Juillet 2005, p. 309-310, Genève, Suisse.

[94] Mockus, Audris, Roy T. Fielding et James Herbsleb, « Two Cases Studies of Open Source Software Development: Apache and Mozilla », *ACM Transactions on Software Engineering and Methodology*, vol. 11, n° 3, 2002, p. 309-346, USA.

[95] Mozilla Firefox, http://www.mozilla.com/fr/firefox/, consulté le 05 juillet 2011.

[96] Niels, Jørgensen, « Putting it all in the trunk: Incremental software engineering in the FreeBSD open source project », *Information Systems Journal*, vol. 11, n° 4, 2001, p. 321–336.

[97] Onofré, Jean-baptiste, « Apache Karaf 2nd Meeting (2012-01-04) », *The Apache Foundation*, https://cwiki.apache.org/KARAF/karaf-2nd-meeting-2012-01-04.html, consulté le 27 Mars 2012

[98] Otte, Tobias, Robert Moreton et Heinz D. Knoell , « Applied Quality Assurance Methods under the Open Source Development Model », *the 32nd Annual IEEE International Conference on Computer Software and Application*, n° 0730-3157, 2008, p. 1247-1252, Turku.

[99] Otto, « Asynchrone », *Dictionnaire Informatique*, http://dictionnaire.phpmyvisites.net/definition-Asynchrone--6910.htm, consulté le 2 Février 2011.

[100] PAC, https://www.pac-online.com/pac/pac/live/pac_world/home/index.html, *Pierre Audoin Consultants*, consulté le 10 Août 2010.

[101] Projet GNU, http://www.gnu.org/philosophy/free-sw.fr.html, consulté le 10 Août 2010.

[102] Raymond, Eric S., *The Cathedral and the Bazaar,* Sebastopol, CA: O'Reilly & Associates, 1999, Canada, http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/, consulté le 14 Août 2010.

[103] Rees, Marc, « Le marché du logiciel libre en France en hausse de 80% », *PC INpact SARL de Presse*, http://www.pcinpact.com/actu/news/34271-marche-libre-logiciel-linux-mandriva.htm, 2007, France, consulté le 10 Août 2010.

[104] Reis, Christian R., et Renata P.d.M. Fortes, « An overview of the software engineering process and tools in the Mozilla project », *CiteSeerX*, 2002, Brésil, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.6127, consulté le 22 Janvier 2011.

[105] Rigby, Peter C., Daniel M. German et Magaret-Anne Storey, « Open source Software Peer Review practices: A Case Study of the Apache Server », *ACM*, n° 978-1-60558-079, 2008, Allemagne.

[106] Sacchi, Walt, « Understanding the Requirements for Developing Open Source Software Systems », *IEEE Proceeding-Software*, vol. 149, n° 1, 2002, p. 24-39, USA.

[107] Scacchi, Walt, « Free and Open Source Development Practices in Game Community », *IEEE Computer Society*, n° 0740-7459, 2004, p. 59-66, USA.

[108] Schmidt, D. C., et A. Porter, « Leveraging open-source communities to improve the quality & performance of open-source software », *Proceedings of the 1st Workshop on Open Source Software Engineering*, 2001, Toronto, Canada.

[109] Séguin, Normand, « Inventaire, analyse et consolidation des principes fondamentaux du génie logiciel », *École de technologies supérieure*, n° 9780494208748, 2006, Montréal, CA, http://proquest.umi.com/pqdweb?did=1251894611& sid=13& Fmt=2& clientId=46962& RQT=309& VName=PQD , consulté le 22 Janvier 2011.

[110] Shibuya, Bianca, et Tetsuo Tamai, « Understand the Process of Participating in Open Source Communities », *Proceedings of the 2009 ICSE Workshop on Emerging Trends in FreeLibreOpen Source Software Research and Development*, IEEE, vol. 0, n° 978-1-4244-3720-7, 2009, Canada.

[111] Site internet sur la norme ISO/IEC 29110, http://profs.etsmtl.ca/claporte/VSE/Groupe24-menu.html, consulté le 23 Avril 2012.

[112] SourceForge, www.sourceforge.net, consulté le 2 Février 2011

[113] Spaeth, Sebastien, Matthias Stuermer, Stefen Haefliger, Georg von Krogh et ETH Zurich, « Sampling in Open Source Software Development: the case for using the Debian GNU/Linux Distribution », *Proceedings of the 40thAnnual Hawaii International Conference on System Sciences, IEEE Computer Society*, n° 0-7695-2755-8, 2007.

[114] Spinellis, Diomidis, Georgios Gousios, Vassilios Karakoidas, Panagiotis Louridas, Paul J. Adams, Ioannis Samoladas et Ioannis Stamelos, « Evaluating the Quality of Open Source Software », *Proceedings of the International Workshop on Software Quality and Maintainability (SQM 2008)*, vol. 233, 2009, p. 5-28, Athens, Greece, texte intégral chez *Electronic Notes in Theoretical Computer Science* à l'adresse: http://www.sciencedirect.com/science/journal/15710661, consulté le 20 Janvier 2011.

[115] Tawileh, Anas, et Omer Rana, « Free and Open Source Software Quality Assurance », *Information and Communication Technologies, IEEE*, vol. 2, 2006, p. 2866 - 2871.

[116] Thiessoz, Yannick, « Méthodologies d'ingénierie logicielle adaptée à une PME », *université de Fribourg*, 2007.

[117] Thomas, « Logiciels libres et propriétaires », *HALPANET*, http://www.halpanet.org/?q=node/11, 2009, consulté le 10 Août 2010.

[118] Ulf, Asklund, et Lars Bendix, « A Study of Configuration Management in the Open Source Software Projects », *IEEE Proceedings Software*, vol. 149, n° 1, 2002, p. 40-46.

[119] Von Krogh, Georg, S. Haefliger et S. Spaeth, « The practice of Knowledge Reuse in Open Source Software: Shifting the creative Effort », *Proceedings of the Academy of Management conference*, 13 Août 2006, Atlanta, USA.

[120] Weber, Steven (2005), « The Success of Open Source », *Cambridge MA: Harvard University Press*, ISBN-10. 0674012925, 2004, publié en ligne dans le *Canadian Journal of Political Science*, vol. 40, n° 1, 2007.

[121] Weigers, Karl, « Creating a Software Engineering Culture », *Dorset House Publishing*, Edition 1, n° 0932633331, 1996.

[122] Xiao-Fang, ZU, et Zheng Jun-hong, « Comparative Discussion on Free software and commercial software Development mode based on CMM theory », *International on Challenges in Environmental Science and Computer Engineering (CESCE)*, vol. 2, 2010, p. 484-487, Chine.

[123] Yamauchi, Yutawa, Makoto Yokosawa, Takeshi Shinohara et Toru Ishida, « Collaboration with Lean Media: How Open-Source Software Succeeds », *Proc. 2000 ACM Conf. Computer Supported Cooperative Work, ACM Press*, 2000, p. 329–338.