MASTER IN COMPUTER SCIENCE

A stakeholder-centric quality model for Open Source Software

Kamseu, Flora

Award date:
2007

Awarding institution:
University of Namur

Link to publication

A *STAKEHOLDER-CENTRIC* QUALITY MODEL FOR OPEN SOURCE SOFTWARE

By

Flora Kamseu

SUBMITTED AS A PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
D.E.A INTERUNIVERSITAIRE EN INFORMATIQUE
AT
UNIVERSITY OF NAMUR
RUE GRANDGAGNAGE 21, NAMUR
AUGUST 2007

UNIVERSITY OF NAMUR
FACULTY OF
COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Computer Science for acceptance a Master entitled "**A *stakeholder-centric* quality model for open source software**" by **Flora Kamseu** in partial fulfillment of the requirements for the degree of **D.E.A Interuniversitaire en Informatique**.

Dated: August 2007

President:	Pierre Yves Schobbens, FUNDP
Research Supervisor:	Naji Habra, FUNDP
External Examiner:	Jean Christophe Deprez, CETIC
Examining Committee:	Jean François Raskin, ULB

UNIVERSITY OF NAMUR

Date: **August 2007**

Author:	**Flora Kamseu**
Title:	**A *stakeholder-centric* quality model for open source software**
Faculty:	**Computer Science**

Degree: **D.E.A** Convocation: **September** Year: **2007**

Signature of Author

To Marthe, Fride,

Noé and Michel

Table of Contents

List of Tables

List of Figures

Abstract

There are several quality models used to evaluate software systems in general; however, none of them is dedicated to Open Source Software applications. The aim of this work is to propose a model for Open Source Software system. The proposed model is a *stakeholder-centric* model.

We also analyze and study several existing software quality models namely : McCall's(Rawashdeh and Matalkah, 2006), ISO Standard (ISO/IEC, 1991), Dromey (Dromey, 1995)and (Dromey, 1996), the QSOS model (QSOS, 2007), the Capgemini Open Source Maturity Model (F. and C., 2003) and the Open Business Readiness Rating (OpenBRR, 2005). We present limitations found in the existing models such as the trend to ignore certain quality feature like functionality or the failure to describe how the quality measurement in these models has been carried out.

Keywords: Open Source Software, quality model, stakeholder, measurement.

Résumé

Il existe plusieurs modèles de qualité pour supporter et structurer l'évaluation des systèmes logiciels, mais aucun de ceux-ci ne s'applique d'une façon satisfaisante à la qualité des logiciels libres. Dans ce travail, nous proposons un model de qualité spécifique pour les logiciels libres. Ce model prend en compte les intérêts des différents types d'utilisateurs présents dans l'environnement du logiciel libre.

Nous commençons par une analyse des principaux modèles de qualité existants notamment les modèles McCall (Rawashdeh and Matalkah, 2006), ISO 9126 Standard (ISO/IEC, 1991), Dromey (Dromey, 1995)and (Dromey, 1996), QSOS (QSOS, 2007), Capgemini Open Source Maturity (F. and C., 2003) et Open Business Readiness Rating (OpenBRR, 2005). Cette analyse nous permet de mettre en évidence les limites de ces différents modèles de qualité pour le logiciel libre. Ces insuffisances portent sur la non prise en compte de certaines qualités tel que la fonctionnalité ou la portabilité.

Mots clés: logiciel libre, modèle de qualité, intérêt des utilisateurs, mesures.

Acknowledgements

I would like to thank Naji Habra, my supervisor, for his many suggestions and constant support during this research. I would also like to thank Pierre Yves Schobbens for his carefully review and suggestions on an earlier version of this report.
I should also mention that my graduate studies in Namur were supported in part by the QualOSS[1] project.
I am grateful to my parents for their love.

Finally, I wish to thank the following:
Clotaire and Ghislaine for their encouragement;
Benoit, Stephane and Ravi for their friendship;

Namur, Rue Grandgagnage 21
Flora Kamseu
August, 2007

[1]QualOSS means Quality of Open Source Software; QualOSS plans to assess the quality of open source projects quantitatively, objectively and rapidly.

Introduction

In the software engineering we can't do without quality. To understand the landscape of software quality it is essential to answer the so often asked question: what is quality? The word *quality* has various definitions. The definition given by the ISO/IEC 8402 standard is : *the totality of features and characteristics of a product or a service that bear on its ability to satisfy stated or implied needs.*

Quality is a multidimensional concept. The dimensions of quality include the entity of interest and the viewpoint on that entity. A popular view of quality is that it is an intangible trait and that it can be discussed and judged, but can not be weighted or measured. From a customer standpoint, quality is the customer's perceived value of the product. The customer purchases or more generally makes a choice, based on a number of variable such as price, performance, reliability, overall satisfaction and others.

The objective of this study report is to introduce a quality model and metrics that help in assessing the quality attributes of an Open Source Software (OSS). The term *stakeholder* is used to refer to any person or group who will be affected by the system, directly or indirectly (Rawashdeh and Matalkah, 2006).

The report is organized as follow : in the chapter 1, we begin with a summary of definitions of quality and related concepts and by introducing the most common and standard quality models and their limitations. After, we summarize a structured synthesis of these quality model by presenting a generic comparison of quality model in chapter 2. Then, the chapter 3 presents briefly the common way to develop an OSS and in chapter 4, we introduce some of the most common characteristics of quality models. They are used to develop a *stakeholder-centric* quality model to assess and measure the quality characteristics that are appropriate for an OSS. Finally, chapter 5 proposes a methodology for the use of the proposed model.

Chapter 1

Software Quality Models State of the Art

There are many standards of quality for software. Some are given by institutions and others by researchers. These standards have common features: the first is the intrinsic product quality and the second is both product quality and customer satisfactions (Kan et al., 1994) . In this chapter, we review definitions of quality an related concepts, we present the most common quality models with their limitations. Precisely, in the next section, we review some definitions of quality, quality model and measurement. It also presents the representation of a quality model. Section 1.2 analyses the main quality models and outlines their limitations for OSS.

1.1 Terminology

1.1.1 Some definitions of quality

Like we said above, the word *quality* has various definitions. In this section, we present the viewpoint of several authors on that concept. It is important to take note about different opinion related to software quality.

Inspired by the technical report wrote by Khashayar Khosravi and Yann-GaÄel (Khosravi and Gueheneuc, 2004), we present some definitions from International and Standard Organizations:

1. ISO 9126: *software quality characteristic is a set of attributes of a software product by which its quality is described and evaluated.*

2. German Industry Standard DIN 55350 Part 11: *quality comprises all characteristics and significant features of a product or an activity which relate to the satisfying of given requirements.*

3. ANSI Standard (ANSI/ASQC A3/1978): *quality is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs.*

4. IEEE Standard (IEEE Std 729-1983):

   (a) *The totality of features and characteristics of a software product that bear on its ability to satisfy given needs.* For example, conformance to specifications.

   (b) *The degree to which software possesses a desired combination of attributes.*

   (c) *The degree to which a customer or a user perceives that a software meets her composite expectations.*

   (d) *The composite characteristics of a software that determine the degree to which the software in use will meet the expectations of the customer.*

Finally, (Rawashdeh and Matalkah, 2006) define quality as *a functional and artistic measurement used for instance, to specify user satisfaction with a product, or how well the product performs compared to similar products.*

1.1.2 Definition and representation of a quality model

A quality model is an abstract form of reality enabling details to be eliminated and an entity or concept to be viewed from a particular perspective. The term *quality model* is defined as *the set of characteristics and relationships between them, which provides the basis for specifying quality requirements and evaluating quality*(ISO, 1999). The main kinds of model are cost estimation models, quality models and maturity models. They are three different ways to represent a quality model: equations, functions or diagrams. In case of the representation of the quality model as a diagram, quality models consist basically of quality characteristics, which are refined into quality sub-characteristics, and finally into measurable properties. We will use this type of representation in the quality model for OSS proposed in chapter 4.

1.1.3 Measurement

As said above, the main objective of a quality model is to operationalize the term quality by making it eventually measurable. For this reason, refinement of quality characteristics inside a quality model stops at the level of quality attributes, which are measurable properties. In order to measure the property or attribute of an entity (i.e., process, product or resource) via measurement, numbers or symbols are assigned to its attribute (Fenton and Pfleeger, 1996) and (Agresti et al., 2002). This assignment has

to follow clearly specified rules so that different people assign the same measurement values to the attribute. A measure is *the number or symbol assigned to a quality attribute of an entity by making a measurement* (ISO/IEC, 1991).
There are direct measures that are determined by directly analyzing the entity under study. Indirect measures, on the other hand, are derived from the measures of one or more other attributes. A descriptive quality model is an operational rule that determines how to measure an entity's attribute. Formally, a descriptive quality model can be defined as a function f that computes a measure $M = f(x_1, ..., x_n)$, from the values of the measures x_i where $i = 1, ..., n$ (Briand et al., 1997).

1.2 Literature study of the main existing quality models

In the literature, several quality models have been defined by different people and organizations. In the following, we summarize briefly some of the most standard and well-known quality models. These quality models could be divided in two categories:

Hierarchical general category which is a group of quality models with representation as a diagram. There are also more general models than the other one and could not be used for an OSS. These representation consists basically of a high characteristics , which are refined into sub-characteristics and finally into measurable properties. These models are described from section 1.2.1 to section 1.2.5.

Category specific for an OSS which are quality models for an OSS. We review these models from section 1.2.7 to section 1.2.9.

We notice that none of the main existing quality models takes into account explicitly the stakeholders interests. Therefore, we propose in chapter 4 a *stakeholder-centric* quality model.

1.2.1 McCall's quality model

Presentation of the McCall's model

Created in 1976, McCall's model for software quality combines *eleven criteria* (Rawashdeh and Matalkah, 2006) around *three stages* of a software lifecycle:

1. Product operations: running and operating;
2. Product revisions: changing and updating;
3. Product transitions: moving to a different context.

These criteria include : correctness, reliability, efficiency, integrity, usability, maintainability, testability, flexibility, portability, reusability and interoperability like show in figure 1.1.

The main idea behind McCall's model is to assess the relationships among external quality factors and product quality criteria. External quality is quality viewed by the customers; Internal quality is quality as measured by the programmer.
McCall's model is used in the United States for very large projects in the military, space, and public domain.
The layers of quality model in McCall are defined as :

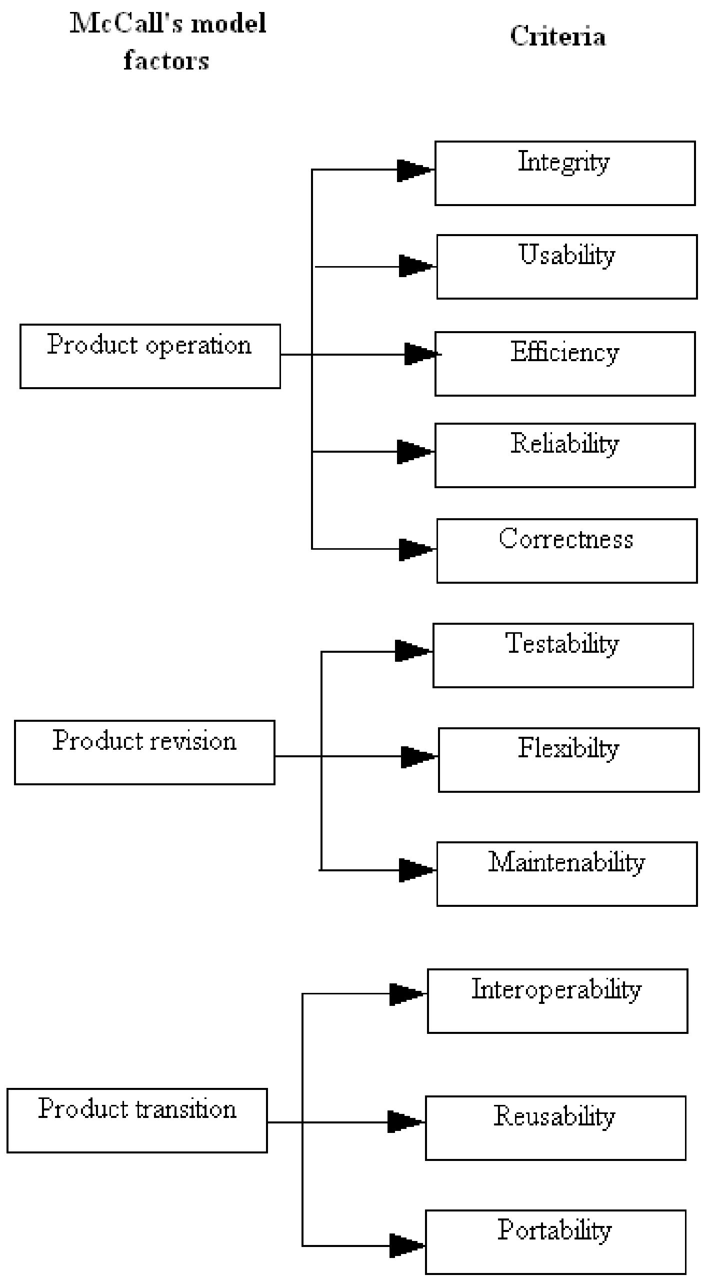


Figure 1.1: McCall's Quality Model

1. Factors which describe the external view of the software, as viewed by the users;
2. Criteria which describe the internal view of the software, as seen by the developer;
3. Metrics which is defined and used to provide a scale and method for measurement.

Contributions and limitations of the McCall model

One of the major contributions of the McCall model is the relationship highlighted between quality characteristics and metrics. Adnan in (Rawashdeh and Matalkah, 2006) criticized that not all metrics are objective. Another limitation is one aspect not considered directly by this model. This aspect is the functionality of the software product.

1.2.2 Boehm's quality model

Presentation of the Boehm model

In 1978, Boehm added some characteristics to McCall's model with emphasis on the *maintainability* of software product. This model also includes considerations involved in the evaluation of a software product with respect to the utility of the program.

In addition, Boehm's model proposed categories attributes according to a *utility view*. The attributes in the model come from three different types of *utility* (see

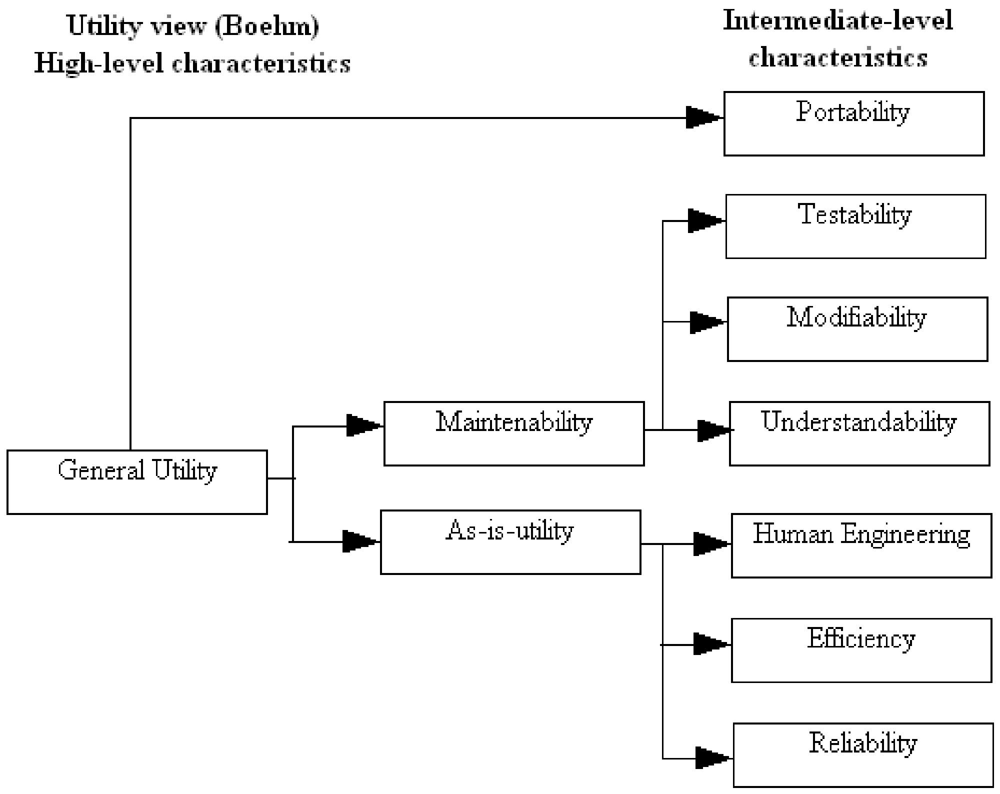


Figure 1.2: Boehm's Quality Model

figure 1.2):

1. As-is utility;
2. Maintainability;
3. Portability.

The layers of quality model in Boehm are defined as (Khosravi and Gueheneuc, 2004):

1. High-level characteristics which represent the general utility of software.
2. Intermediate-level characteristics which the Boehm's 7 quality factors that together represent the qualities expected from a software system.

3. Primitive characteristics which provide the foundation for defining qualities metrics.

Contributions and limitations of the Boehm model

Like expressed in (Ortega et al., 2003), the Boehm quality model is similar to the McCall quality model in that it represents a hierarchical structure of characteristics, each of which contributes to total quality. Boehm's notion includes users needs, as McCall's does. However, it also adds the hardware product characteristics not encountered in the McCall model. Boehm's quality model contains only a diagram to organize into a hierarchy characteristics without any suggestion about measuring those characteristics.

1.2.3 FURPS quality model

Presentation of the FURPS quality model

The FURPS model was proposed in 1987 by Robert Grady and Hewlett-Packard Compagny. The FURPS model takes into account the following five characteristics that make up its name:

- Functionality
- Usability
- Reliability
- Performance

- Supportability

The FURPS-categories are of two different types:

- Functional requirements (F): defined by input and expected output;
- Non-functional requirements (URPS): Usability, Reliability, Performance, Supportability.

Contributions and limitations of the FURPS model

One disadvantage of this model is that it fails to take account of the software product's portability (Ortega et al., 2003).

1.2.4 ISO/IEC 9126 Quality Model

Presentation of the ISO/IEC 9126 quality model

With the need for the software industry to standardize the evaluation of software products using quality models, the ISO (International Organization for Standardization) proposed in 1991 a standard (ISO/IEC, 1991) which specifies six areas of importance for software evaluation and, for each area, specifications that attempt to make the six area measurable. The layers of quality model in ISO/IEC are defined as show in figure 1.3.

- External and internal quality: *internal quality* is the totality of characteristics of a software product from an internal view, while an *external quality* is the totality of characteristics of the software product from an external view (ISO/IEC,

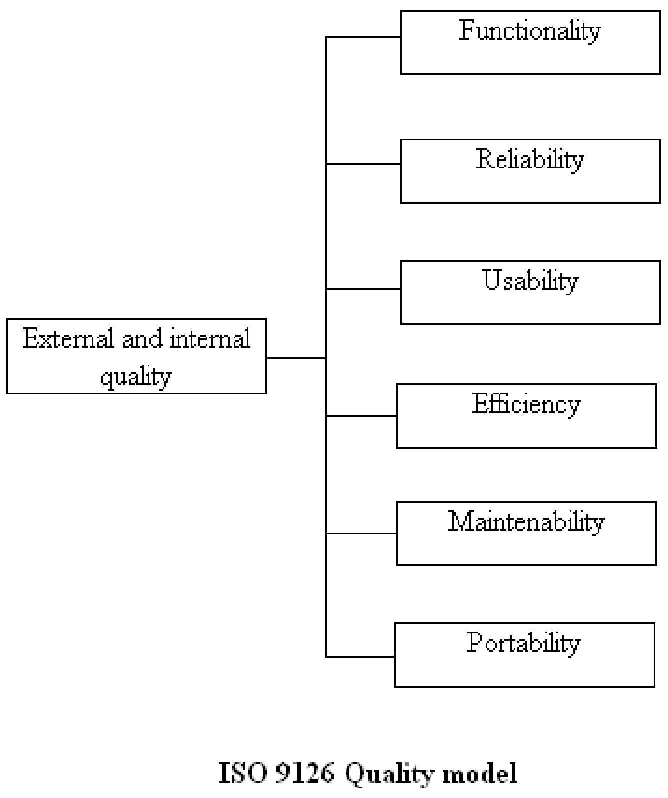


Figure 1.3: ISO/IEC 9126 Quality Model

1991). The *internal attribute* can be observed without any execution of the software and thus with no connection with environmental, while the *external attribute* are observed during execution of the software with environment (Habra et al., 2007)

- Characteristics (functionality, reliability, usability, efficiency, maintainability, portability).
- Sub-characteristics.
- Metrics.

Each characteristic is refined to a set of sub-characteristics and each sub-characteristic is evaluated by a set of metrics.

Contributions and limitations of the ISO/IEC 9126 model

One of the advantages of the ISO 9126 model is that it identifies and distinguishes the internal characteristics and external quality characteristics of a software product. ISO 9126 is also recognized as a standard model.
However, at the same time it has the disadvantage of not showing very clearly how these characteristics can be measured (Ortega et al., 2003).

1.2.5 Geoff Dromey's quality Model

Presentation of the Dromey quality model

In 1996, Dromey proposed a new quality model having same similarities with the McCall, the Boehm's and the FURPS quality models. Dromey suggests a framework

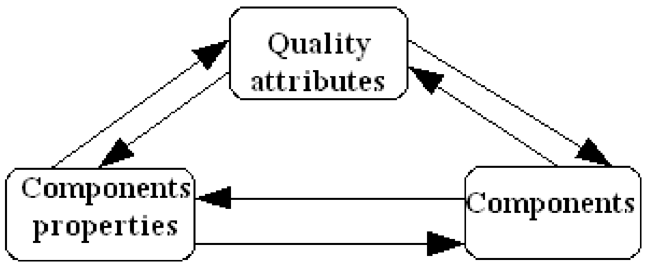


Figure 1.4: Factors that determine quality product in Dromey's model

for the construction, the use of practical, testable quality model for requirements, design and implementation (Dromey, 1995)and (Dromey, 1996). In referring to the well-known expression *build quality into software*, Dromey points out that high-level quality attributes, such as reliability and maintainability, cannot be built into the software. What can be done though is to identify a set of properties (such as modules without side effects) and build them up consistently, harmoniously and fully to provide reliability and maintainability. Links must be forged between the tangible properties of the product and the high-level quality attributes (Ortega et al., 2003). Dromey proposes three models, depending on the products resulting from each stage of the development process: requirements quality model, design quality model, and implementation quality model (programming). It define factors that determine a quality product (see figure 1.4):

- A set of components ;
- A set of quality-carrying properties of components;
- A set of high quality attributes.

Dromey insists in the fact that we need a quality process to produce a quality product (Dromey, 1996) . Dromey identifies five steps to build his model:

1. Choose a set of high-level *attributes* that we need to use for evaluation.
2. Make a list of all the components or modules in the system.
3. Identify *quality-carrying properties* for each component that is, *qualities* of the component that has the most impact on the product properties from the list created in last step.
4. Decide on how each property affects the quality attributes.
5. Evaluate the model.
6. Identify and resolve weaknesses in with feedback loop.

The quality model of Dromey' product implementation is represent by figure 1.5.

Contributions and limitations of the Dromey model

This model adds reusability of software to the international standard ISO 9126 software product evaluation. This model raises a number of important issues about programming language design. For example, most existing languages leave the responsibility for satisfying the various *quality-carrying properties* in the hands of the designer or the programmer.

Dromey suggests that programmers have to change their style of implementation and/or submit their programs to much more rigorous compiler checks which insist that quality requirements are satisfied before a compiler will produce executable code.

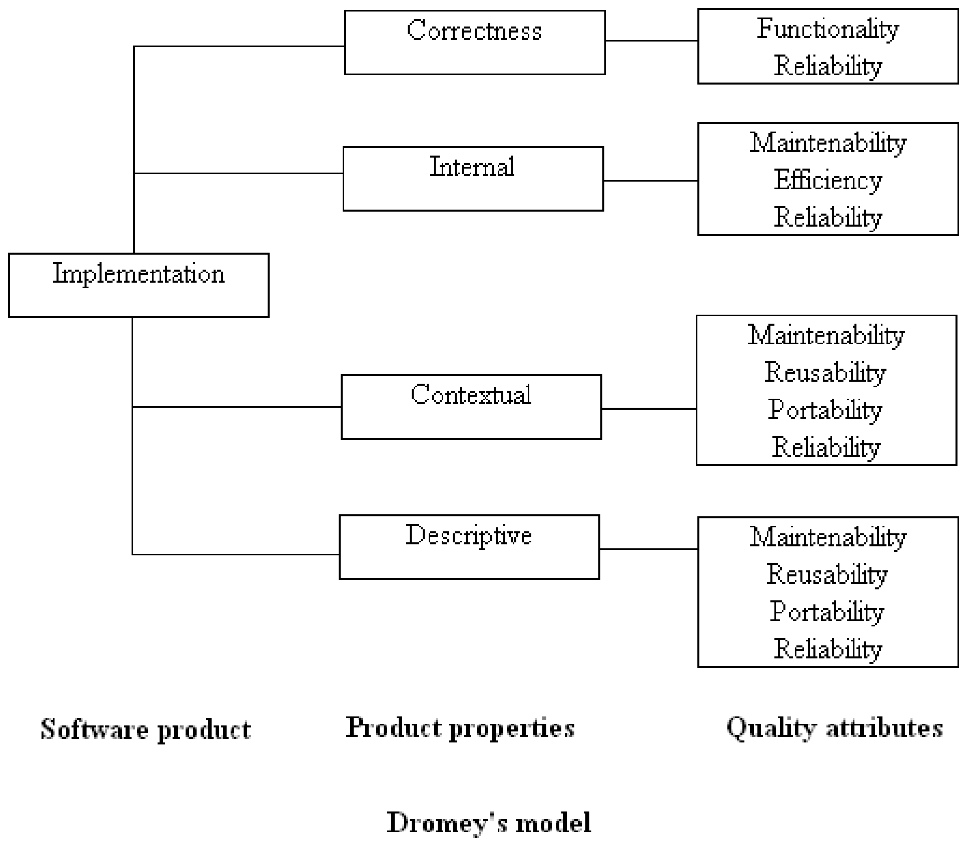


Figure 1.5: Dromey Quality Model

The disadvantage of the Dromey model is associated with reliability and maintainability. For the author, it is not feasible to judge both attribute (reliability and maintainability) of a system before it is actually operational in the production area.

1.2.6 ISO 9126 VS McCall quality model

As show on figure 1.6, there are many similarities between characteristics in the McCall quality model and the ISO/IEC 9126 quality model.

1.2.7 QSOS model

Presentation of the QSOS model

QSOS (QSOS, 2007) split its evaluation template in two kinds of sections: one generic section and several sections specific to a particular family of applications such as Groupware, CMS, Database,....

Contributions and limitations of the QSOS model

In some case, the terminology used by QSOS is not detailed enough. References is not precise enough as its is unclear to the reader to know what characteristic of reference are under consideration.

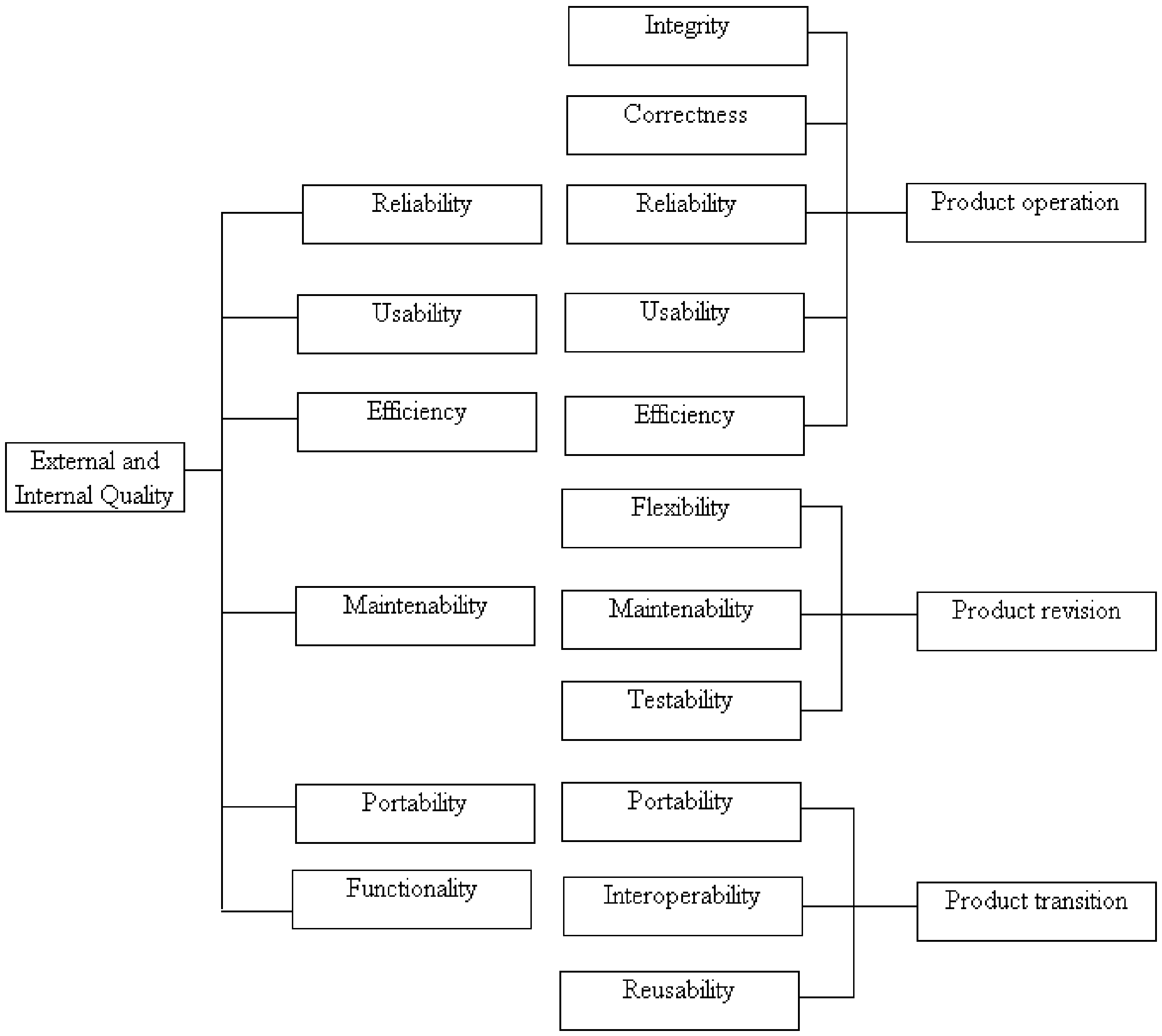


Figure 1.6: ISO/IEC 9126 VS McCall Quality Model

1.2.8 Open Source Maturity (OSM) model

Presentation of the OSM model

Below is the hierarchy proposed by OSMM, as defined by Cap Gemini (F. and C., 2003):

- Product
  - Age
  - Licensing
  - Human hierarchies
  - Selling points
  - Developer community
- Integration
  - Modularity
  - Collaboration with others products
  - Standards
- Use
  - Support
  - Ease of deployment
- Acceptance
  - User community
  - Market penetration

Contributions and limitations of the OSM model

The hierarchy is much lighter and it is also very imprecise in its language. For example, *Integration .. Standards* does not specify what characteristics related to standards are under consideration.

1.2.9 Open Business Readiness Rating (OBRR) model

Presentation of the OBRR model

The OpenBRR builds on two existing general maturity models, Navica's Open Source Maturity Model (Golden, 2005) and Cap Gemini's equivalent (OpenBRR, 2005). The OpenBRR hierarchy of quality characteristics looks as follows:

- Usability
  - End user UI experience
  - Time for setup pre-requisites for installing open source software
  - Time for vanilla installation/configuration
- Quality
  - Number of minor releases in past 12 months
  - Number of point/patch releases in past 12 months
  - Number of open bugs for the last 6 months
  - Number of bugs fixed in last 6 months
  - Number of P1/critical bugs opened

  - Average bug age for P1 in last 6 months

- Security
  - Number of security vulnerabilities in the last 6 months that are moderately to extremely critical
  - Number of security vulnerabilities still open (unpatched)
  - Is there a dedicated information (web page, wiki, etc) for security?

- Performance
  - Performance Testing and Benchmark Reports available
  - Performance Tuning and Configuration

- Scalability
  - Reference deployment
  - Designed for scalability

- Architecture
  - Is there any 3rd party Plug-ins
  - Public API / External Service
  - Enable/disable features through configuration

- Support
  - Average volume of general mailing list in the last 6 months
  - Quality of professional support

- Documentation
  - Existence of various documentations
  - User contribution framework
- Adoption
  - How many books does amazon.com gives for Power Search query: *subject:computer and title:component name*
  - Reference deployment
- Community
  - Average volume of general mailing list in the last 6 months
  - Number of unique code contributor in the last 6 months
- Professionalism
  - Project Driver
  - Difficulty to enter the core developer team

Contributions and limitations of the OBRR model

Unlike the other two previous hierarchies, elements of OpenBRR hierarchy are very specific, for example, under the *Quality* category, we find *number of minor releases in the past 12 months.* This is not a quality characteristic but as OpenBRR calls it: a *metric.*

1.3 Conclusion

Several models, including hierarchy and OSS' models specializing in measuring the quality of software products have been described. These models have been studied, analyzed and their limitations outlined. Specifically, functionality of a software product was not considered directly by McCall's model. No suggestion about measuring the quality characteristics has been found in Boehm's model. FURPS model fails to take account of the software product's portability. ISO 9126 has the limitation of not show very clearly how certain quality aspects can be measured. The disadvantage of Dromey's model is associated with reliability and maintainability. It is not possible to judge these two attributes of a system before it is actually operational in the production area. The disadvantage of OSS' models (QSOS, OSMM and OBRR) are : the terminology used by QSOS is not detailed enough , the hierarchy of OSMM is much lighter and it is also very imprecise in it language and elements of OpenBRR hierarchy are very specific. The study of the OSS' models shows that they are not mature enough (Kamseu and Habra, 2007). The specific models for an OSS are recent, not really test or used. They are specific for industry and their industrial occupancy is limited.

Among all the existing models that have been studied, we found that ISO 9126 is the most complete model with some limitations. Our proposed model will be based on some quality characteristics of this model.

Chapter 2

Generic comparison of quality models

This chapter combines different layers of the hierarchical[1] quality models studied in chapter 1 to construct a mapping between the key concepts of the main quality model. This work would outlines the similarities between these concepts. In the next section, we present layers from the hierarchical quality model. Theses layers are used to construct a mapping in section 2.2.

2.1 The concept of *layer* in the main quality models

Models studied in chapter 1 have different layers. We present and describe these layers per model. This section present the definition and description of terms from layers in main quality models used to construct a mapping between these models.

McCall's quality model : it has three layers.

1. Factors (to specify): they describe the external view of the software, as viewed by the users. The quality factors describe different types of system behavioral characteristics. They can have many criteria.

[1]We restrict ourself on hierarchical models: McCall, Boehm, Dromey and ISO 9126 models.

2. Criteria (to build): they describe the internal view of the software, as seen by the developer. They can have many metrics.
3. Metrics (to control): they are defined and used to provide a scale and method for measurement. The quality metric, in turn, aims to capture some of the aspects of a quality criteria.

Boehm's quality model : it has four layers.

1. High-level characteristics : they represent basic high-level requirements of actual use to which evaluation of software quality could be put (the general utility of software). The high-level characteristics address three main questions that a buyer of software has:
   - As-is utility: How well (easily, reliably, efficiently) can I use it as-is?
   - Maintainability: How easy is it to understand, modify and retest?
   - Portability: Can I still use it if I change my environment?
2. Intermediate level characteristic : it represents Boehm's 7 quality factors that together represent the qualities expected from a software system :portability, reliability, efficiency, usability, testability, understandability, flexibility.
3. Primitive characteristics metrics hierarchy: they provide the foundation for defining qualities metrics.
4. Metrics

ISO/IEC 9126 quality model : it has three layers.

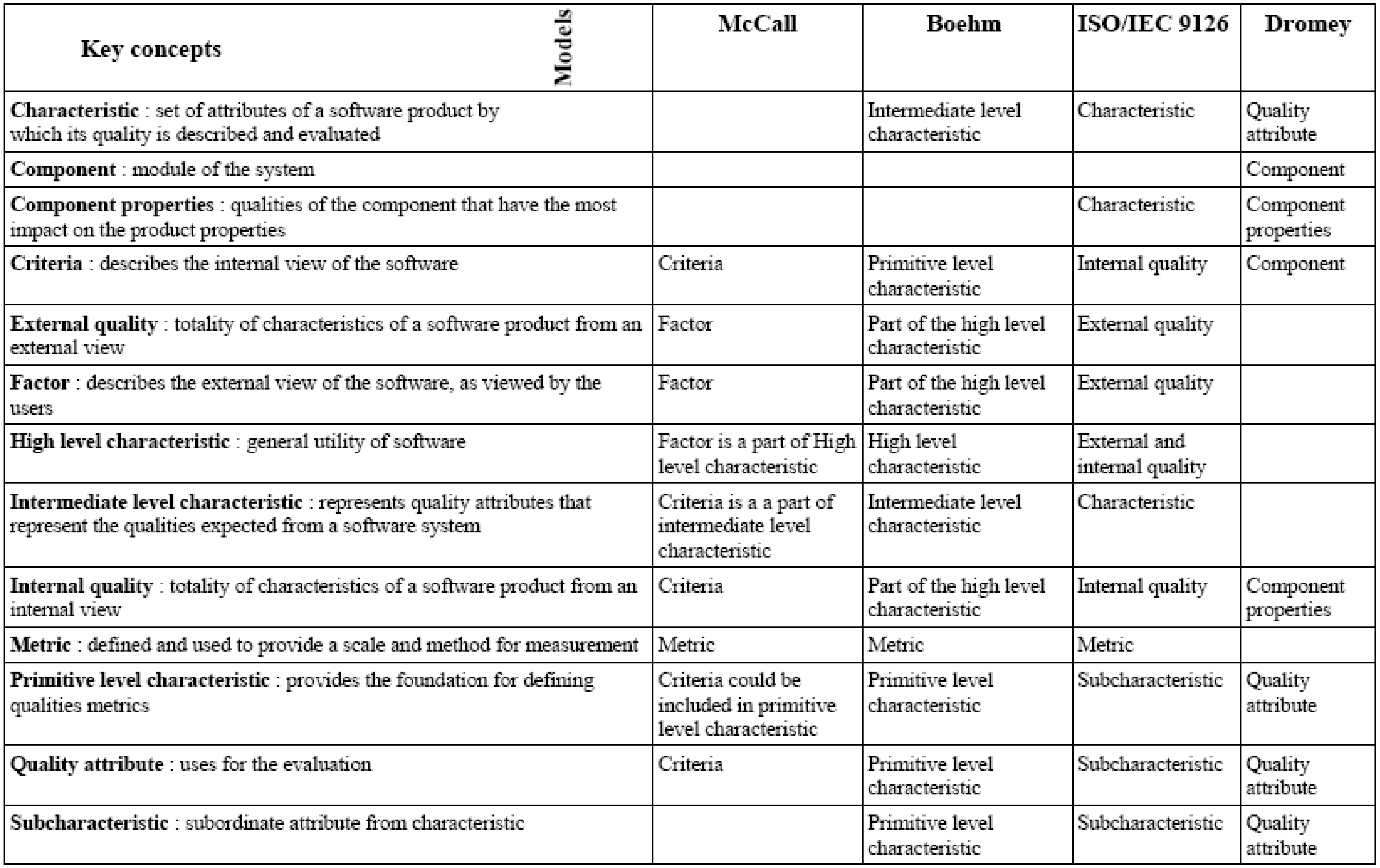
1. Characteristics: as given in ISO 9126, *software quality characteristics is a set of attributes of a software product by which its quality is described and evaluated.* This set of attributes includes functionality, reliability, usability, efficiency, maintainability, and portability.
2. Sub-characteristics.
3. Metrics

Dromey quality model : it has three layers.

1. Quality attributes necessary for the evaluation.
2. Components or modules of the system.
3. Components properties for the components/modules. That are qualities of the component that have the most impact on the product properties.

2.2 Mapping between the key concepts from the hierarchical quality models

The figure 2.1 shows different relationship between layers from the main quality models.



Key concepts / **Models**	**McCall**	**Boehm**	**ISO/IEC 9126**	**Dromey**
Characteristic : set of attributes of a software product by which its quality is described and evaluated		Intermediate level characteristic	Characteristic	Quality attribute
Component : module of the system				Component
Component properties : qualities of the component that have the most impact on the product properties			Characteristic	Component properties
Criteria : describes the internal view of the software	Criteria	Primitive level characteristic	Internal quality	Component
External quality : totality of characteristics of a software product from an external view	Factor	Part of the high level characteristic	External quality	
Factor : describes the external view of the software, as viewed by the users	Factor	Part of the high level characteristic	External quality	
High level characteristic : general utility of software	Factor is a part of High level characteristic	High level characteristic	External and internal quality	
Intermediate level characteristic : represents quality attributes that represent the qualities expected from a software system	Criteria is a a part of intermediate level characteristic	Intermediate level characteristic	Characteristic	
Internal quality : totality of characteristics of a software product from an internal view	Criteria	Part of the high level characteristic	Internal quality	Component properties
Metric : defined and used to provide a scale and method for measurement	Metric	Metric	Metric	
Primitive level characteristic : provides the foundation for defining qualities metrics	Criteria could be included in primitive level characteristic	Primitive level characteristic	Subcharacteristic	Quality attribute
Quality attribute : uses for the evaluation	Criteria	Primitive level characteristic	Subcharacteristic	Quality attribute
Subcharacteristic : subordinate attribute from characteristic		Primitive level characteristic	Subcharacteristic	Quality attribute

Figure 2.1: Mapping between the key concepts from the hierarchical quality models

Chapter 3

Development process of OSS

OSS is computer software whose source code is available under a copyright license that allows users to study, change, and improve the software, and to redistribute it in modified or unmodified form. One research question of interest is how OSS projects work? How are they organized? What methods and techniques do they use to produce software?

As a *product*, an OSS is a software like all others softwares: a sequence of instructions to be interpreted by a computer, which performs actions accordingly. OSS exists in all shapes and sizes and is made to solve different tasks. Most software solves a specific problem. For example word processing software solves the problem of writing text, creating layout for the text; Email clients are software for sending and receiving emails, and the list goes on. Software helps people perform different tasks, some of which can only be performed by use of a computer with the proper software. The difference between OSS and proprietary software consists in the type of licence and then to the development process. The next section describes the type of licence in OSS and section 3.2 presents the development process of an OSS

3.1 The type of Licence in OSS

OSS is computer software, which comes with a license that is different from the licenses in commercial software such as Microsoft Office suite or the Windows Operating System. The license is a legal agreement between the user and the producer. The license defines the terms of use, that a user must accept to be allowed to use the software. Commercial companies have typically relied on very restrictive licence schemes, which allowed the user a minimum of rights. Commercial licenses do not allow users to copy or modify the software in any way. Users are only allowed to use the software. Commercial companies never release the source code for their software. OSS exists since the early days of computing as software sharing. The name OSS indicates, one of the special properties of the license: the source code for the software is freely available–open source. Apart from access to the source code, OSS licenses have other properties, which are very different from that of the usual commercial software license.
An OSS license grants the users (Kumiyo et al., 2002):

Access to source code : access to source code is required, and any person creating OSS has to make the source code for his software available to anyone.

The right to freely redistribute : the right to freely redistribute means that any person, who accepts the license, is allowed to make as many copies as he wishes and distribute these copies.

The right to create derived work : the right to create derived work allows anyone to use the source code from OSS, modify the source code and distribute this work under a new name. It is, however, required that reference the original contributor are made. The original contributor retains copyright for the code that he wrote.

The obligation to redistribute the licence : to keep the software from changing license to a non-open source license, it is required that the same license is used for distribution. A person making copies of OSS is therefore required to copy the license.

An OSS license is clearly very different from its commercial counterparts. None of the mentioned four properties are ever granted in commercial software licenses. We should stress that OSS should not be confused with *free software.* The term ***free software*** does not mean non-commercial. *Free software* is a matter of the users' freedom to run, copy, distribute, study, change and improve the software (Limi et al., 2007). Similarly, OSS should not be confused with shared source software (the source code of which is visible, but there are limitations on use, modification, or redistribution).

This report uses the term ***open source*** for its usual meaning, that is, software which has its source code freely available for use, viewing, modification, and redistribution.

3.2 Development process of an open source software

OSS exists in all sorts and sizes, and every day new OSS projects are initiated. It is important to emphasize that the nature of the activities in OSS projects is a development effort.

3.2.1 Different roles of members in an OSS community

Members of the most OSS project assume certain roles by themselves according to their personal interest in the project (Kumiyo et al., 2002). A member may have one of the following eight roles:

1. **Project Leader** : it is usually the person who has initiated the project. This person is responsible for the vision and overall direction of the project. He is often the person who conceived the idea for the software.

2. **Core member** : a core member is responsible for guiding and coordinating the development major points of an OSS project. He has been involved with the project for a relatively long time and has significant contributions to the development and evolution of the system.

3. **Active developers** : they are people who regularly contribute new features and fix bugs. Together, they contribute one of the major development forces of OSS systems.

4. **Peripheral developer** : they contribute occasionally to create new functionalities or features to the existing system. Their contribution is irregular and the period of involvement is short and sporadic.

5. **Bug fixer** : they fix a bug, that they either discovered themselves or was reported by bug reporters. Bug fixers have to read and understand a small portion of the source code of the system where the bug occur.

6. **Bug reporter** : they discover and report bugs. They do not fix the bugs themselves and they may not read source code either. They play the same role

as testers of the traditional software development organization.

7. **Readers** : they are active users of the system. They not only use the system, but they also try to understand how the system works by reading the source code. Readers are like peer reviewers in a traditional software development organization.

8. **Passive User** : they are those who just use the system in the same way as most of those use commercial software. They are attracted to OSS mainly due to its high quality and the potential of being changed when needed.

The first two groups (project leader and core members) are also called maintainers. The maintainer takes on a special responsibility for the project and functions as the personal point of contact for the project. The maintainer is the person who releases new and improved versions of the software. As such, the maintainer has the final word on what features and suggestions should be incorporated into the software. All people who participate in the project are referred to as *contributors*. Figure 3.1 presents members' contribution for developing an OSS.

3.2.2 Communication between members in an OSS community

A common feature of most OSS development projects is the means of communications employed. Communications in OSS development are maintained primarily using services facilitated by the Internet such as email, newsgroups, mailing lists, web pages, and chat (Kamseu and Habra, 2007). Of the mentioned forms of communication, chat is the only real-time medium.

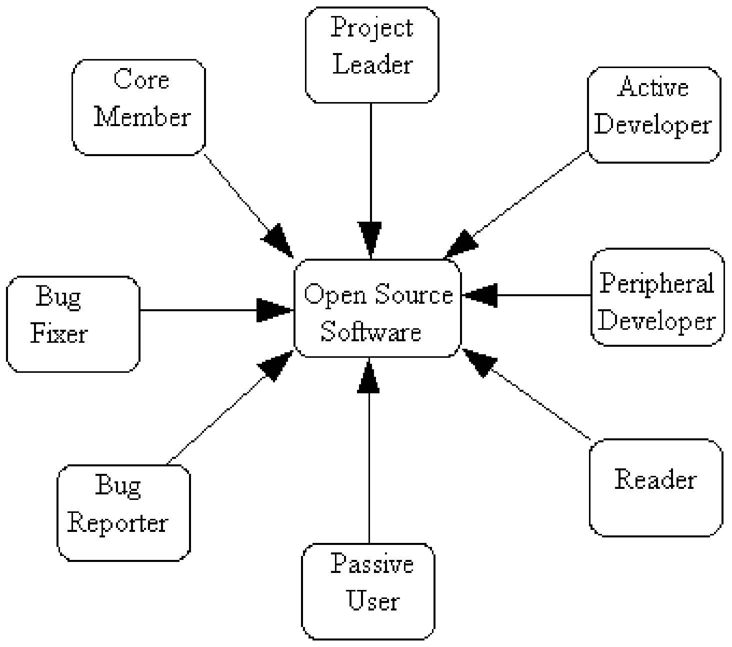


Figure 3.1: Members contributing for an OSS quality software

Occasionally telephone is being used, but this is a rare event mostly due to the cost of long distance phone calls and the intimate nature of personal phone calls. FreeBSD developer Poul-Henning Kamp noted that phone calls were invaluable when trying to solve a complex problem, but did require a level of personal intimacy not usually associated with OSS development (Edwards, 2001). Phone calls also moved the development process out of the regular forum and isolated the discussion from comments. Most projects use a web page for general information and downloading the project software. Central means of communications are facilitated by mailing lists, which provide a centralized way of reaching all the people in the project.

3.2.3 Lifecycle of activities for an OSS product

As presented above, the development of an OSS is different from the commercial one. The basic development cycle of an OSS development can be schematized as follows:

1. }Previous steps ...
2. Maintainer releases software and source code
3. Contributors download software and source code
4. Contributors identify problems or needed features
5. Contributors implement corrections
6. Corrections are emailed to the Maintainer/mailing list for inclusion in the project
7. Corrections are discussed on the mailing list
8. Maintainer reviews the corrections and includes changes

1. Maintainer releases new version software and source code
2. Contributor downloads software and ... and so aford

3.2.4 A typical process

The maintainer initiates the projects by making available the first version of the software on the project home page and making announcements on various news groups, mailing lists and home pages. Interested people download the software to try it. Some people immediately dislike the software, others find it useful and begin to use the software. Some of those who find the software useful also identify problems or needed features in the software. Interested people then subscribe to the project mailing list to receive discussions regarding the project and its further development. Given the ability to code and the source code, the interested person now begins to make changes to the software. The interested person is now becoming a contributor to the project. Once changes are made, the contributor submits the changes (a patch) to the project mailing list and/or the maintainer. It is the hope of the contributor that his patch will be included in the project. The maintainer and/or other persons on the mailing list will review the submitted patch and discuss it. The interest in the submitted patch is correlated to the interest that other people have in that particular area of the software. Often the code will not be discussed in detail, simply because no one on the list is interested in that part of the software. Following review and perhaps some discussion, the patch is either accepted or rejected. If accepted the patch is included in the software, and the new version is made available for downloading. Often several or even hundreds of persons are engaged in a development project and contribute to the best of their efforts. Large variations in contributed code can be observed in the projects.

Participation in development requires a person to read or scan a lot of emails referred to as high traffic. The Linux kernel development is an example of a project with a high traffic mailing list. In week 20 of year 2001 the Linux kernel mailing list received 1227 posts from 423 different contributors 45% posted more than once, and 38% posted in the week before (Edwards, 2001). This suggests that active persons on the list frequent the list. The amount of emails to the list is enough to discourage many people.

The actual source code development takes place in two spheres: at home and in the mailing list. Contributors work in their homes and develop the software using their personal computer. The source code they produce are later submitted to the project mailing list. Suggestions are made and backed up with technical arguments. Often there is a back and forth discussion where different suggestions are tested privately and the results are discussed. The testing is a central part of the development, and since system configuration and usage pattern vary among users, it is important to find a solution, which satisfies most people.

We can summarize these activities with their relationships to the stakeholder contributor in figure 3.2.

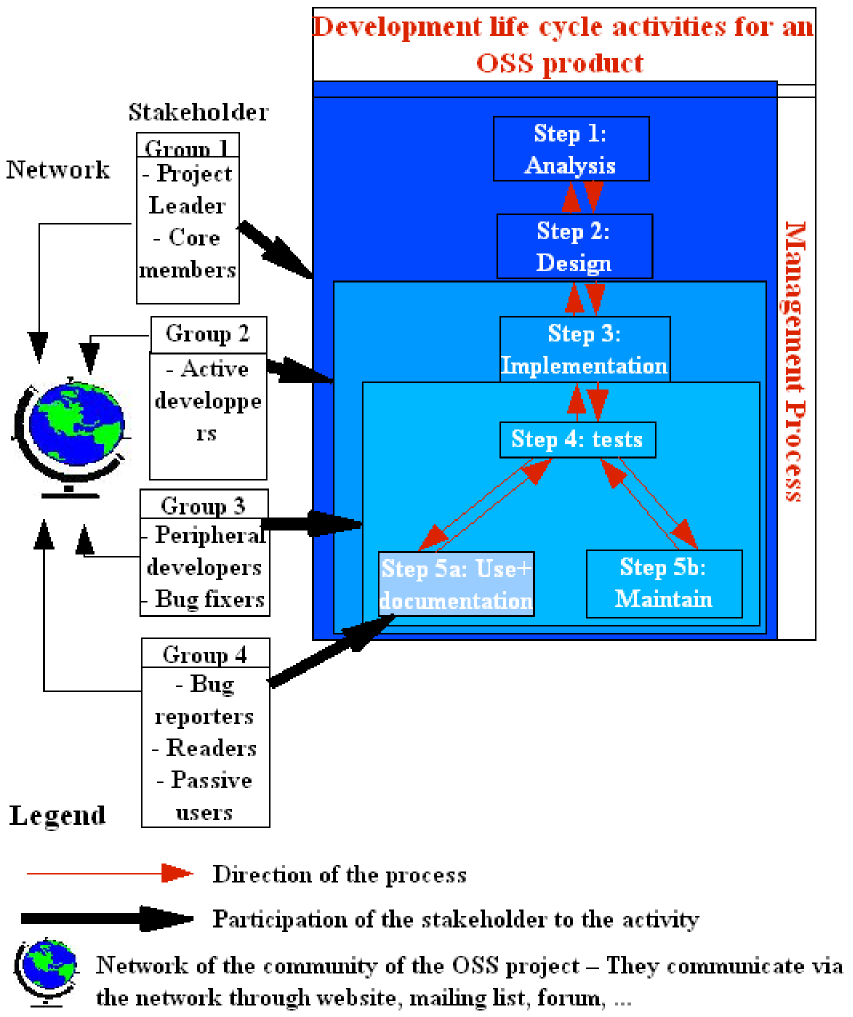


Figure 3.2: Activities life cycle for an OSS product

Chapter 4

A *stakeholder-centric* quality model for OSS

This chapter aims at building a quality model to improve productivity and usage of OSS. To this end, it starts by listing features that we think are relevant to assess and measure an OSS through a quality model. These quality attributes are then used to develop a *stakeholder-centric* quality model for an OSS. Our aim is to provide an understandable and a useful framework to evaluate the strength of OSS. This framework will meet specific stakeholders quality needs for an OSS application. Precisely, in section 4.3, we provide a method in four step to design our quality model, which we then construct in section 4.3.4.

Introduction

OSS development and use has increased significantly over recent years (Raja and Barry, 2005). Software quality is an important issue having an impact to overall system lifecycle cost, performance and useful life. Research on proprietary software or Closed Source Software (CSS) has revealed that software quality declines over time

(Raja and Barry, 2005). Part of this decline is associated with the lifecycle maintenance activities that increase continously in the size and complexity of the system. Lifecycle maintenance activities in OSS systems are processed under a very different context. The OSS movement is changing the way software is developed, maintained and updated (Wheeler, 2006). Therefore, quality issues perspectives change also.
In this research, we explore some important software characteristics that contribute to consistent OSS quality. In the following sections, we present quality characteristics and the different type of stakeholders in OSS. We then develop a model for quality in OSS. The proposed quality model is built on basis of this stakeholder typology and takes into account stakeholders interests and function. We used this approach because there are a number of different interest groups who have quite distinct software product quality requirements. Our model will need to properly accommodate these requirements.

4.1 Terminology and framework

A lot of questions appears in discussions about software product quality because there is a confusion about how various terms are used. That is why we first introduce our terminology before starting the construction of the model for OSS.

- Some of the properties of software are *desirable*. We call these desirable properties *qualities* or *quality attributes*. Quality or more specifically a set of quality attributes is the vehicle through which the different interest groups express their requirements.

- They are two types of quality attributes of software, those corresponding to the *behaviors* of software and those corresponding to its *uses.* A *behavior* is something that the software itself *exhibits* when it executes under the influence of a set of inputs.

- A *use* is something that different interested groups *do* with or *to* software.
- To accommodate and balance the needs of different interested group in building a quality model, we rely on a *Goal Directed Approach* (Dromey, 1998). *A Goal Directed Approach to building a quality model for software is effective for accommodating and balancing the needs of these interest groups* (Dromey, 1998). We will use this approach to build our quality model.

4.2 Requirements to build an OSS quality model

According to *Goal Directed Approach*, three issues must be addressed to formulate the requirements for a software quality model (Dromey, 1998):

1. The different *interest groups* need to be identified.
2. The *intended applications* of the model need to be spelled out.
3. It is necessary *to establish the quality needs of the different interest groups.*

A constructive strategy can be used to characterize the behaviors and uses of OSS that contribute to its quality. We will employ decomposition to characterize or define *behaviors* or *uses* in terms of subordinate *behaviors*, *uses* and software characteristics. The two principles that guide decomposition are :

Principle 1 : A *behavior* can be decomposed and defined in terms of subordinate properties which may be described as behavior or as a software characteristic.

Principle 2 : A *use* can be decomposed and defined in terms of subordinate properties which may be described as uses or software characteristics.

4.3 Construction of a quality model for an OSS

In this section, we proceed to the construction of a quality model for an OSS. We use a four step modeling as outlined below :

1. Identification of the different types of stakeholders in OSS. This is done in section 4.3.1.
2. Identification and description of the quality attributes in OSS. We describe quality attributes in section 4.3.2.
3. Description of some principles and guidelines for the architecture or structure of the design of the quality model (see section 4.3.3).
4. Effective construction of the quality model for OSS. We propose a general scheme of *stakeholder-centric* quality model. This model is applied in an OSS community in section 4.3.4.

4.3.1 Different types of stakeholders in OSS

The aim is to identify all the stakeholders in OSS. One example of different roles in an OSS community is described in section 3.2.1. We can divide an OSS community into two groups:

Group 1 : persons who contribute to the development and evolution of the OSS by using the source code of the software. These contributions might be: adding new features or fixing bugs, reading the source code and reporting some comments.

Group 2 : those who are not interested in the source code. They use the system in the same way as most of those who use proprietary software. They can contribute by discovering and reporting bugs, providing comments on the mailing list (see section 3.2.2).

4.3.2 Quality attributes in OSS

Software quality is one of the most important indicators for the success of a software project.

Categorization : the first task in building a software product quality model is to identify what the intended applications of the model are and to address the needs of the different interest groups that will use the software model. This influences the quality attributes to focus on. We can divide OSS into two types: (1) projects that are developed to reproduce and replace existing Closed Source Software (CSS) and (2) projects initiated to create new software that has no existing equivalent in CSS.

Identification of the quality attributes : We identify a set of important quality attributes in OSS of the two categories (Raja and Barry, 2005) which are described below.

A) Functionality which refers to providing minimum functions as required by the user. This quality attribute is concerned with what the software does to fulfill the identified needs. For type-1 OSS, there are no explicit functionality requirements. There will be a certain level of expectations in term of its functionality compared to an existing CSS. New user will adopt type-1 software if it provides the basic functionality of its CSS equivalent.

In case of type-2 OSS, there is no existing software to derive functional requirements from. Therefore, new users will be defining such requirements according to their own needs. In type-1, OSS will be considered as of high quality if it provides the basic functionality of its CSS equivalent. On the other hand, type-2 OSS will be considered of a high quality if it provides the functional requirements of its active users. Functionality can be decomposed into three sub-quality attributes:

A-1) Suitability which reflects the capability of a software product to provide an appropriate set of functions for specified tasks and user objectives (ISO/IEC, 1991).

A-2) Interoperability that is the capability of a software product to meaningfully interact with others software (ISO/IEC, 1991).

A-3) Security that is the capability of the software to protect data from unauthorized persons. In OSS, the software source code is available globally. This means that users can identify potential vulnerabilities. It also implies that pirates can exploit these vulnerability easily. The quality will depend on how vigilant the active users detect vulnerabilities and protecting the software from bad intention. In that case, more the OSS will be secure, more the software will be consider as high quality.

B) Reliability is the ability of a system to perform its required functions for a period of time (IEEE, 1990). The reliability is concerned with the behavior of the software. Ideally, the software should behave as expected in type of possible states of the environment. Although OSS is available free of cost, these software needs to have a minimum operational reliability to make it useful for any application. Reliability has a significant effect on software quality, since the user acceptability of a product depends on its ability to function correctly and reliably. Reliability can be decomposed in :

- **B-1) Fault tolerance** which is the capability of a software to have a specific level of performance even under the occurrence of a fault.
- **B-2) Maturity** which is the capability of a software to avoid failure and faults.
- **B-3) Recoverability** reflects the length of time and effort a software takes to recover from a failure.

C) Availability means that the software should be available to the user during a proportion of time. It means that the software should be functional at that time.

D) Reusability : in OSS, there is usually no estimates of development or maintenance cost (Raja and Barry, 2005). OSS communities encourage development and use of reusable modules that can be shared. OSS that employs reusable modules will attract more contributions and maintain a high quality. Reusability can be decomposed in :

D-1) Application independence represents the separation of the application from the system where it is installed.

D-2) Representation independence is the separation of representation of data from the programs that use the data.

D-3) Data encapsulation which is sometimes referred to as data hiding, is the mechanism whereby the implementation details of a class are kept hidden from the class user.

D-4) Function encapsulation is a mechanism used to kept a function hidden from the class user.

D-5) Interface ability is how easily a user can interact with the software using an interface.

E) Maintainability in general refers to the ability to maintain the system over a period of time. This will include ease of detecting, isolating and removing defects. Additionally, factors such as ease of addition of new functionality, interface to new components, programmers ability to understand existing code and test team's ability to test the system will increase the maintainability of the system. OSS is downloaded and use by the a global community of users. There are usually no face to face meetings among the users of the software. They have to rely on the documentation in the source code and on the communication through forum messages. OSS is required to be highly maintainable. Because participation is voluntary, low maintainability will generate minimum participation of active users and then, will have a negative effect on quality. Maintainability can be decomposed in :

E-1) Analysability is the capability to easily analyse the source code and architecture of a software.

E-2) Learnability is the capability to easily understand existing code and to learn how to use a software.

E-3) Modifiability is the degree to which the software facilitates the incorporation of changes (Boehm and W, 1978) such as new functionality, interface to new components to the system.

E-4) Testability is the degree to which the software facilitates the establishment of tests (IEEE, 1990).

F) Portability is the capability of the software product to be transferred from one environnement to another. Portability can be decomposed in :

F-1) Installability represents the capability of a software product to be installed in a specified environment (ISO/IEC, 1991).

F-2) Machine independence represents the independence of the software application from machine.

F-3) System independence is how independent is the software product from the operating system that interact with the computer at a very basic level.

F-3) Repleaceability is the capability of a software product to be used in place of another specified software product for the same purpose in the same environment (ISO/IEC, 1991).

4.3.3 Description of some principles and guidelines

The third step for building this model is to identify a suitable architecture for the model.

In proceeding to construct an OSS quality model, we will employ the following additional design principles, guidelines and assumptions:

- We choose to associate a target of quality attributes with OSS.
- The quality of software may be characterized by the set of high quality attributes.
- Each quality attribute of a given software correspond either to a set of domain independent *behaviors* of software and/or a set of domain independent *uses* of software.
- The chosen quality attributes of the proposed quality model should be sufficient to meet the needs of all interest groups associated with the software.
- Each high level quality attribute of software is characterized by a set of subordinate properties which are either behaviors, uses.

In section 4.3.4, we provide a generic quality model (see figure 4.1) and an instantiation of this quality model for an OSS (see figure 4.2. We start out with the principal interest groups, then we look for what are their quality requirements. Matching with the different interest groups is what we call a *stakeholder-centric* quality model.

4.3.4 Proposed quality model for OSS

We describe in figure 4.1 the general scheme for a *stakeholder-centric* model.

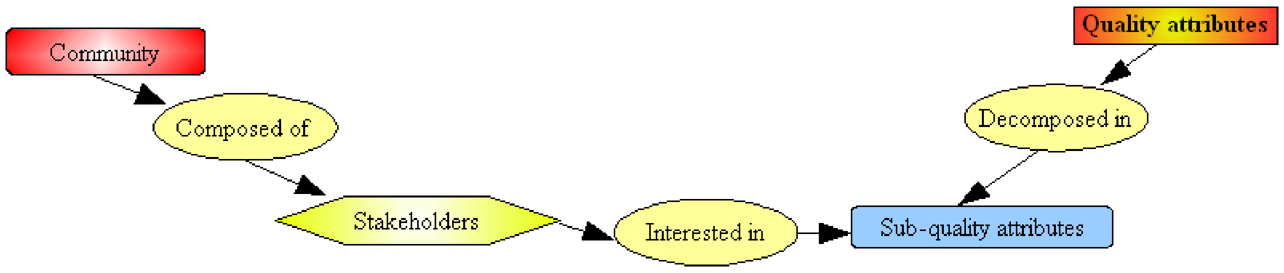


Figure 4.1: General scheme for stakeholder centric model

Figure 4.2 represents our proposed *stakeholder-centric* quality model specific for an OSS. We show the different interests of every type of user from the quality attributes. This model is an instance of the general scheme for *stakeholder-centric* model.

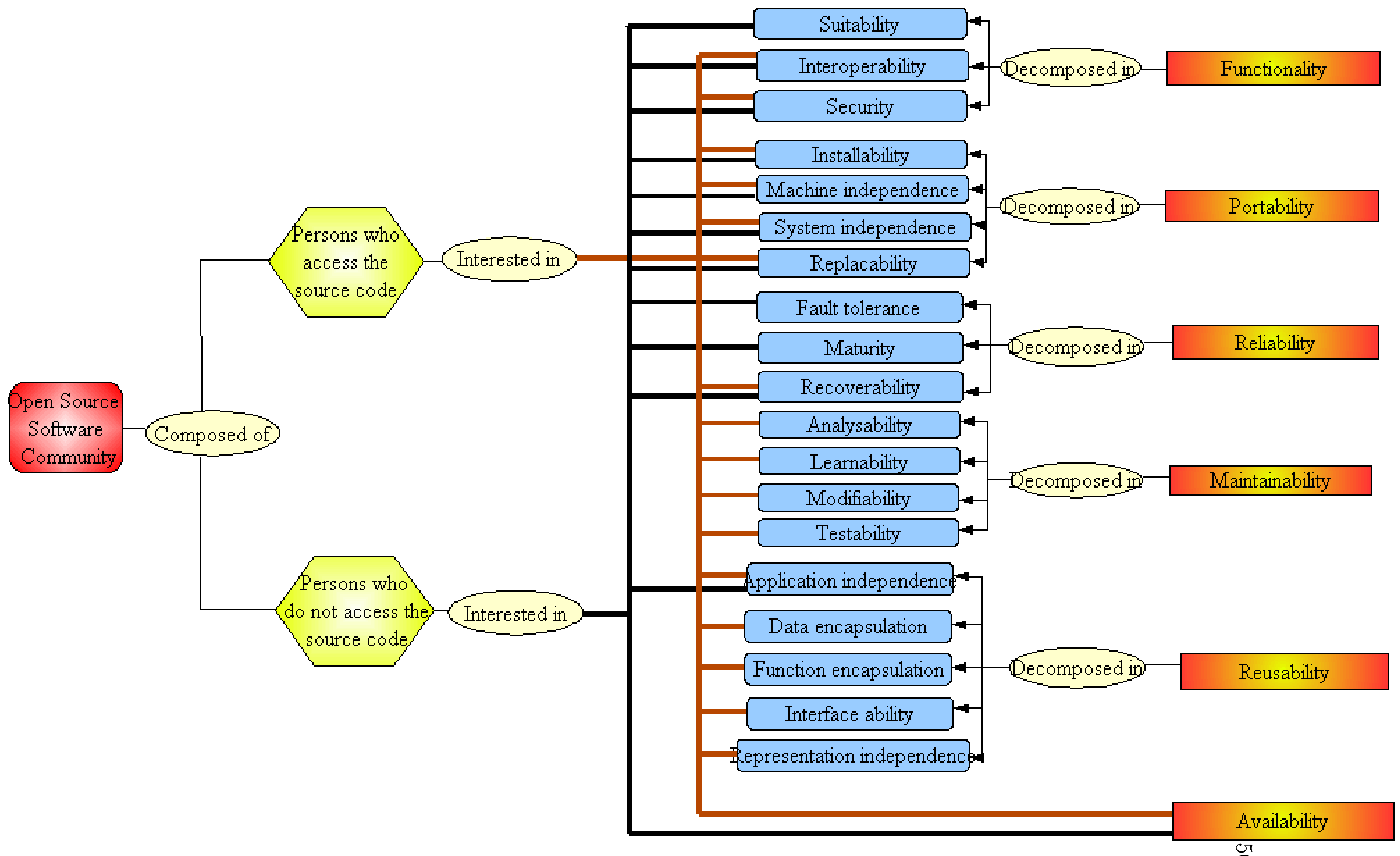


Figure 4.2: Proposed *stakeholder-centric* quality model for OSS

4.4 Similarities and differences with others quality models

Many similarities and also differences appear between the proposed model and others quality models.

- The proposed quality model is a hierarchical one like those reviewed in chapter 1.
- We have included quality attributes that reflect considerations for type of use in an OSS environment. Those quality attributes are functionality, portability, reliability, maintainability, reusability and availability.
- We have kept many similarities with ISO 9126 (ISO/IEC, 1991) namely the definition of quality attributes apart from the inclusion of availability as a top-level quality attributes. This is because without the source code availability, users can not access the source code. We have also add reusability as a quality attribute because of the interest of stakeholders (Raja and Barry, 2005);
- A new set of sub-quality attributes has been defined and associated with each high level quality attribute.
- Quality models like this are never absolute or fixed, either in terms of the chosen primary interest groups or in terms of the set of high quality attributes. The point we are trying to make here is that quality needs vary according to the context but a general classification is still applicable.

- The framework we are proposing is robust because it takes into account different concerns and needs from every user interact with the system. It is also flexible enough to accommodate variability, change and refinement because of the different type of needs.

4.5 Conclusion

This model has been developed using identification of characteristics that meets the needs of different interest groups involved with OSS (Raja and Barry, 2005).
A significant advantage of the proposed model is that it takes into account the different needs of different users in OSS. The number of OSS systems continues to increase significantly. The need for a model that ensures quality attributes of such system become a necessity. Therefore, the proposed model could be use to train or educate people to produce a quality software and facilitate the uses of software.

Chapter 5

Methodology for the use of the model

In this chapter, we propose some methods for use the proposed *stakeholder-centric* quality model. The next section introduces the *Goal Question Metric* (G.Q.M) approach which is used to develop the methodology to use the proposed model. Section 5.2 outlines some questions concerning the proposed model we introduce a discussion about metrics in section 5.3.

Goal Question Metric

In chapter 4, we proposed a *stakeholder-centric* quality model for OSS. This model takes accounts for the interests of different types of stakeholder in terms of what qualities they need for their product. The proposed model is dynamic and flexible in the sense that it may be adjusted according to the type of stakeholder interests. In our proposed quality model, we distinguished two groups of stakeholders and matched sub-qualities to each group.

In this chapter, we suggest how to use the proposed quality model. This method uses the GQM approach (Basili et al., 1992). The result of the application of a GQM approach is the specification of a measurement system with a particular set of issues.

The GQM model is a hierarchical structure starting with a goal which is refined into several questions which are refined into metrics. It has three level (Basili et al., 1992):

1. Conceptual level (**Goal**): a goal is defined for an object, for a variety of reasons, with respect to various points of view, relative to a particular environment.
2. Operational level (**Question**): a set of questions is used to characterize the way the achievement of a specific goal is going to be performed based on some characterizing model.
3. Quantitative level (**Metric**): a set of data is associated with every question in order to answer it in a quantitative way.

The next section presents the methodology and section 5.3 suggest a discussion about the assessment of this model.

5.1 Goal : generic

In this work, we focus on the two first steps of the GQM approach : the ***goal*** and the ***question***. The proposed model has two main goals :

Goal 1 : define the different kind of persons involved in the model.

Goal 2 : define and describe the quality attributes which interest these persons (stakeholder interests).

5.2 Questions : examples

After the definition of these main goals, the next step will be to make an interview by asking questions to the persons concerned. An example of these types of questions is proposed in table 5.1.

Interview Guide
Goal 1 **: define the different kind of persons involved in the model**
Question 1 : who are involved in the organization ?
Question 2 : why do they use this software ?
Question 3 : what do they plan to do with this application ?
Question 4 : who are the people involved in the project ?
Question 5 : why did they choose this software product ?
Goal 2 **: define and describe the quality attributes which interest these persons (stakeholder interests)**
Question 1 : in what kind of application are they interested in ?
Question 2 : what kind of interest do they have with this application ?
Question 3 : what are their needs ?
Question 4 : for how long do they want to use the application ?
Question 5 : define and describe the quality attributes they need for the application

Table 5.1: Examples of questions for the two goals

It will be interesting to group the different types of stakeholders depending on their interests and their needs. The next step will be to construct the quality model. It this a first step to choose or implement an OSS product which fulfill the needs of the different types of stakeholders.

5.3 Metric : discussions

In this section, we want to introduce some dicussion about the assessment of the proposed quality model. This section describes how we intend to aggregate metrics of sub-quality attributes. Most of the quality attributes are refined into sub-quality attributes, the idea is to assign a concrete indicator to interpret the model. Quality attributes are of different importance depending on the interest of the stakeholder. This will be represented through weights. The stakeholder will have to be able to adjust these weights. Aggregation in this view basically consists of calculating the weighted means of the sub-quality attributes.
The objective is to give a quantitative indicator to any metric derived from the sub-quality attributes. Currently, we have not identified those metrics and it will be the objective of a future work. This will allow us to generate an overall quantitative indicator of an OSS. To define this quantitative indicator, lets F_i be the quality attributes with i =$1, .., n$, where n represents the number of the quality attributes. Lets F_{ij} be the sub-quality attribute of each i with j=$1, ..., k_i$, where k_i represents the number of the sub-quality attributes. Note that n and k_i depend both on the stakeholder and correspond to the number of attributes he is interested in. If we denote by M_{ij} a measure of F_{ij}, a measure of F_i can be defined by $F_i =$

$f_i(M_{i1}, M_{i2}, ..., M_{ik})$ where f_i is a $R^{ki} \rightarrow R$ function. An aggregate indicative value of the quality of an OSS will be defined by $F = f(F_1, ..., Fn)$ where f is a function of $R^n \rightarrow R$.

For example, the proposed quality model will have i=1, ..., 6 and F_1=functionality, F_2=portability, F_3=reliability, F_4=maintenability, F_5=reusability, F_6=availability. The aggregate value for the proposed quality model is F=$[F_1; F_2; F_3; F_4; F_5; F_6]$.

Conclusion and future work

In this report, the aim was to propose a *stakeholder-centric* quality model for OSS. In chapter 1, we reviewed the literature concerning the quality and precisely the quality model. We then present the main quality models and their limitations. Chapter 2 proposed an ontology of the main hierarchical quality models. Chapter 3 showed the principle of an OSS project and how the community interacts in those project. This type of communication enable us to propose a *stakeholder-centric* quality model for OSS in chapter 4. Then, the chapter 5 showed a methodology for the use of the model. It also introduce some guidelines to assess and measure the quality attributes.

Although the research area of OSS is relatively young, documented research, contributions are coming at a rapid speed. Overall, there is still much to be learned about the open source model. With this effort, we have started to qualify this model and its potential. However, further studies are necessary to provide additional empirical evidence. We plan to continue our research in this area by collecting metrics of interest to assess the proposed quality model. Currently, we have not yet identified authoritative and alternate formulae to do it. This can be accomplished in future research work which will try to quantified the proposed model by using the GQM approach. This can be done through interviews and analyzes of empirical data.

Bibliography

Agresti, W. W., Basili, V. R., Caldiera, G., and Rombach, H. D. (2002). *Encyclopedia of Software Engineering. Volume 1., ch. Measurement.* John Wiley Sons.

Basili, V. R., Gianluigi, C., and Rombach, H. D. (1992). The goal question metric approach.

Boehm and W, B. (1978). *Characteristics of software quality*, volume 1 of *TRW series of Software technology.* Amsterdam, New York, Oxford.

Briand, L. C., Differding, C., and Rombach, H. D. (1997). Practical guidelines for measurement-based process improvement. *Software Process Improvement and Practice Journal*, (3).

Dromey, R. G. (1995). A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2):146–162.

Dromey, R. G. (1996). Cornering the chimera. *IEEE Software*, 13(1):33–43.

Dromey, R. G. (1998). Software product quality: Theory, model and practice. *Software Quality Institute.* `http://www.sqi.gu.edu.au/docs/sqi/misc/SPQ-Theory.pdf`.

Edwards, K. (2001). Epistemic communities, situated learning and open source software development. `citeseer.ist.psu.edu/edwards01epistemic.html`.

F., D. and C., W. (2003). Open source maturity model, cap gemini. `http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf`.

Fenton, N. E. and Pfleeger, S. L. (1996). *Software Metrics - A Practical and Rigorous Approach*. International Thomson Computer Press. pages 219–242.

Golden, B. (2005). *Succeeding with Open Source*. Addison-Wesley. Making Open Source Ready for the Enterprise: the Open Source Maturity Model, Navica White Paper.

Habra, N., Abran, A., Lopez, M., and Sellami, A. (2007). A framework for the design and verification of software measurement methods. *Journal of Systems and Software, Special Issue on Software Process and Product Measurement, Elsevier*.

IEEE (1990). *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. Institute of Electrical and Electronics Engineers.

ISO (1999). ISO/IEC 14598 international standard, standard for information technology - software product evaluation - part 1: General overview. *IEEE*.

ISO/IEC (1991). Software product evaluation and quality characteristics and guidelines for their use. *ISO/IEC Standard 9126*.

Kamseu, F. and Habra, N. (2007). Etude de la maintenabilité et de l'évolutivité du logiciel libre: cas de PhpMyAdmin et du serveur Apache. *AIM 2007*.

Kan, S. H., Basili, V. R., and Shapiro, L. N. (1994). Software quality: an overview from the perspective of total quality management. *IBM SYSTEMS JOURNAL*, 33(1).

Khosravi, K. and Gueheneuc, Y.-G. (2004). A quality model for design patterns. Technical report 1249; DIRO, university of Montreal.

Kumiyo, N., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y. (2002). Evolution patterns of open source software systems and communities. *ACM 1-58113-545*.

Limi, A., Runyan, A., and Andersen, V. (2007). The free software definition. Free Software Foundation. `http://www.fsf.org/licensing/essays/free-sw.html`.

OpenBRR (2005). Business readiness rating for open source: a proposed open standard to facilitate assessment and adoption of open source software. `http://www.openbrr.org/wiki/images/d/da/BRR_whitepaper_2005RFC1.pdf`.

Ortega, M., Perez, M. A., and Rojas, T. (2003). Construction of a systemic quality model for evaluating software products. `http://www.lisi.usb.ve/publicaciones/`.

QSOS, P. (2007). Method for qualification and selection of open source software. `http://www.qsos.org`.

Raja, U. and Barry, E. (2005). Investigating quality in large-scale open source software. *Proceedings of the fifth workshop on open source software engineering, ACM Press*, pages 1–4.

Rawashdeh, A. and Matalkah, B. (2006). A new software quality model for evaluating COTS components. *Journal of Computer Science 2*, 4:373–381.

Wheeler, D. A. (2006). Why open source software / free software (oss/fs, floss, or foss)? look at the numbers! `http://www.dwheeler.com/oss_fs_why.html`.