Software Quality Model for Consumer Electronics Product

Chanwook Kim, Keun Lee
Samsung Advanced Institute of Technology
Samsung Electronics co., Ltd.
{chanwook.kim, gskeun.lee}@samsung.com

Abstract

Software quality in consumer electronics product is an important factor determining the quality of whole product. Software quality can be evaluated in various aspects depending on domain's characteristics. Therefore, it is very important to identify quality factors reflecting domain's characteristics to evaluate quality of software precisely. Currently, consumer electronics domain demands lower development cost and faster time-to-market. So, code reuse is considered as a general solution satisfying such demand. Therefore, software quality models need to be customized in view of reusability as quality goal. In this paper, we have identified some quality characteristics of ISO 9126 as critical quality factors for consumer electronics software. Also we have introduced metrics related to critical quality factors. Derived quality model can be utilized for both quality evaluation and improvement of consumer electronics software.

1. Introduction

The software in consumer electronics (CE) product, such as digital TV and mobile phone has increased significantly in recent years, hence the quality of software is getting more critical. Quality can be defined as the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs [1]. The needs which should be satisfied by a software product are variable according to the domain's characteristics. It means that the software quality is evaluated based on each domain's characteristic. Therefore, it is very important to identify CE domain's characteristics and related quality attributes to evaluate quality of CE product software.

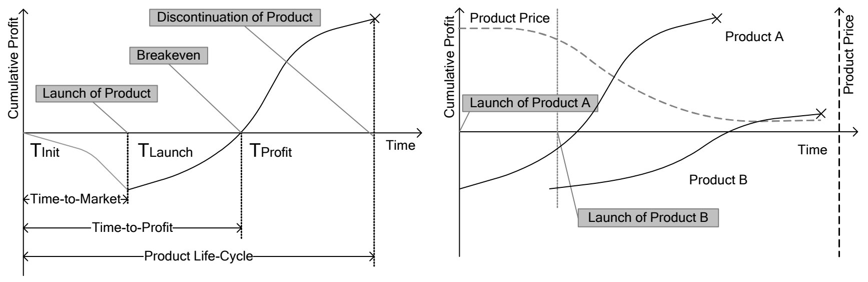
Earlier software quality models such as McCall's model [2] and ISO 9126 quality model [1] identified characteristics for software quality, and they provided how to evaluate the quality characteristics. But, they are not enough to evaluate software quality in CE product, because they lack a rationale for determining which factors should be included in the quality definition [3]. Also, may not every quality characteristics influence software quality equally. Therefore, the quality characteristics have to be prioritized according to the CE domain's characteristics. CE domain's characteristics should be considered to prioritize quality characteristics are as follows:

Firstly, profit made on a CE product depends on the time-to-market. Currently, so many companies are competing against each other in CE market to increase their market share. So, price of product is dropping sharply and product life cycle is getting short. Figure 1- (a) describes the trend of cost and profit over time. As soon as a CE product comes out, its price tends downward. In early state, the profit grows up rapidly, but the profit growth gets slow soon, finally it fades out from the market. As figure 1-(b) shown, if two or more products are sharing same market competitively, the trend is more remarkable, so that, products come out too late will take low profit relatively. Therefore, market gives CE companies a strong pressure for development period reduction.

Also, as the number of product models in same product family is increased, total cost for software developments is growing up rapidly. Currently, many CE companies concurrently release many products which are different in their price, functionality and performance according to consumers' requirements. The growth of product models leads to the increase of development cost, and it has a bad impact on the profit. Therefore, CE companies need to reduce cost for software development.

Generally, it is not easy to satisfy such two conflicting demands, reduction of development period and cost at once. Nevertheless, it is possible to satisfy them at once if productivity of developers is highly improved. Therefore, many self-respecting companies are trying to improve developers' productivity through

1550-6002/09 $26.00 © 2009 IEEE
DOI 10.1109/QSIC.2009.58



(a) Trend of cost and profit over time (b) Trend of cumulative profit in competition

Figure 1. Trend of profit and cost depends on time-to-market

code reuse strategy. Component based development, and software product-line engineering are widely accepted strategies for code reuse. [9-11] That's why what we identify reusability as quality goal in CE domain.

In this paper, we determine weight for each quality characteristic in ISO 9126, based on reusability as quality goal for CE domain, and identify quality characteristics with high priority as critical quality factors. Also, we introduce some useful metrics for critical quality factors.

The rest of this paper is structured as follows: Background researches are reviewed in section 2. Section 3 describes the critical quality factors for CE domain, and how they were identified. Also, the relationship between metrics and critical quality factors is described. In next section, the customized model is applied to a software module as an evaluation. Finally, a conclusion and future works are given in section 5.

2. Background Researches

Since late 70's, many researchers have studied software quality. They often have built quality models such as McCall's model, Boehm's model, and ISO 9126 to understand and measure quality [1-4]. The quality models are defined as a set of characteristics and relationships between them. Therefore, software quality can be specified and evaluated by measuring the quality characteristics.

Even though the earlier quality models are useful; they are not enough. They are too general, and quality goal for any specific domain is not considered. In this section, we introduce well-known quality models briefly, and describe their disadvantage.

2.1. McCall's Quality Model

One of the earliest quality models was suggested by Jim McCall and his colleagues. They defined software quality as a hierarchy of factors, criteria, and metrics. A quality factor represents a behavioral characteristic of the system, as viewed by the users. A quality criterion is an attribute of a quality factor that is related to software production and design, as seen by the developers. A quality metric is a measure that captures some aspect of a quality criterion. The evaluation of quality metrics contributes to quality criteria and, quality factors can be measured by quality criteria. Thus, the 11 quality factors contribute to a complete software quality. Metrics, the bottom of the quality model, are derived from the number of "yes" responses to questions related to each metric.

Even though McCall's quality model is useful, it has a weak point. Metrics composing bottom of the model depend on a number of questions, which can be answered with yes or no. Therefore, it is not an objective way to evaluate software quality. [5]

2.2. ISO 9126

One of the most widely accepted quality model has been introduced by the International Standardization Organization (ISO). ISO 9126 specifies six characteristics for internal and external quality, which are further divided into subcharacteristics. These subcharacteristics are manifested externally when the software is used as a part of a computer system, and are a result of internal software attributes. [1] Figure 2 shows the structure of ISO 9126 quality model. Follows are descriptions for each characteristic:

- Functionality: Functionality is "a set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs".

- Reliability: Reliability is a set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
- Usability: Usability is a set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of uses.
- Efficiency: Efficiency is a set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
- Maintainability: Maintainability is a set of attributes that bear on the effort needed to make specified modifications.
- Portability: Portability is a set of attributes that bear on the ability of software to be transferred from one environment to another.

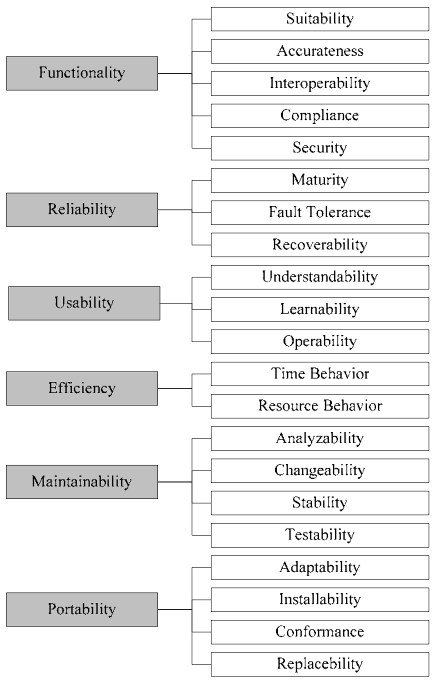


Figure 2. ISO 9126 Software Quality Model

Even though the quality characteristics in ISO 9126 are well defined, they are not specific. The quality model is too generic to apply on a CE product software evaluation. Hence, we need to customize it based on CE domain's characteristics.

3. Software Quality in CE Product

We have accepted ISO 9126 being used widely for quality evaluation as our base model. The six characteristics specified in ISO 9126 mean every attributes the general software should satisfy. But, it does not mean that every software should satisfy every quality attributes equally always. The priority for each characteristic depends on quality goal which should be satisfied by the software, and quality goal is influenced by characteristics of domain. Therefore, quality goal should be identified based on domain characteristics, and priority for each quality characteristic need to be given. Then, we can identify the critical quality factors for CE product software.

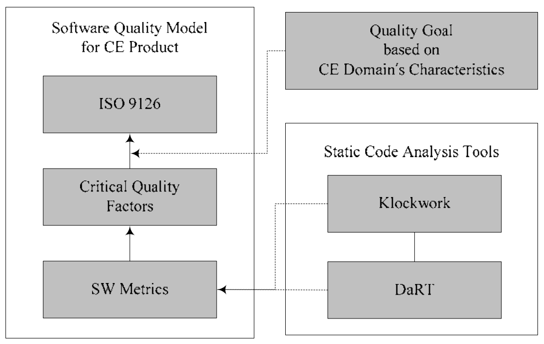

Software Quality Model for CE Product
ISO 9126
Critical Quality Factors
SW Metrics
Quality Goal based on CE Domain's Characteristics
Static Code Analysis Tools
Klockwork
DaRT

Figure 3. Overview of Software Quality Model for CE Product

The quality characteristics described at top of the quality hierarchy are abstract concepts specifying quality of software, so they cannot be measured directly. Each quality characteristic is realized with related sub-characteristics, and each sub-characteristic can be measured by related metrics. Therefore, identifying metrics describing each sub-characteristic is very important to evaluate software quality correctly.

In this section, we specify how we identified critical quality factors for CE product software and selected metrics for quality factors. Figure 3 is an overview of our approach.

3.1. Identifying Critical Quality Factors for CE Product Software

As we described in section 1, according to characteristics of CE domain, pursuing reusability is an important quality goal for CE product software. However, code reuse does not satisfy the quality goal always. Code reuse needs cost and effort. Sometimes, code reuse requires more cost and effort than developing new one. Therefore, it is very important to get source codes, which can be reused for other products with a small amount of cost and effort. Reusability can be defined as a set of attributes that bear on the effort and cost needed to reuse source code for other product environment.

There are two ways to deploy the concept of reusability to quality model. First is adding reusability as a new quality characteristic. We can add it only when the reusability is independent of the other quality characteristics mentioned in ISO 9126. But, reusability depends on six characteristics directly or indirectly, so it cannot be added as a new quality characteristic. An alternative way is determining weight for each existing quality characteristic based on the relationship with reusability. Our approach is based on it.

3.2. Prioritizing Quality Characteristics

It is not easy to determine relative weight for each quality characteristic based on relationship to reusability. With Analytic Hierarchy Process (AHP), we have calculated weight for each characteristic. AHP is one of the most appropriate methods for weighting the criteria [6]. It is a strong managerial tool for multi-criteria decision making.

Table1. Prioritizing Quality Characteristics based on AHP

(a) Comparison Matrix for Quality Characteristics

	F	R	U	E	M	P
F	1.00	0.38	1.59	1.26	0.44	0.33
R	2.62	1.00	3.00	2.62	0.79	0.38
U	0.63	0.33	1.00	0.63	0.38	0.33
E	0.79	0.38	1.59	1.00	0.44	0.33
M	2.29	1.26	2.6	2.29	1.00	0.63
P	3.00	2.62	3.00	3.00	1.59	1.00

F: Functionality, R: Reliability, U: Usability,
E: Efficiency, M: Maintainability, P: Portability
CI = 0.023 CR = 0.019

(b) Weights for Quality Characteristics

Quality Factor	Weight	Normalized Weight
Functionality	29.0791	0.0986
Reliability	**58.7511**	**0.1993**
Usability	21.3710	0.0725
Efficiency	26.8697	0.0912
Maintainability	**62.6566**	**0.2126**
Portability	**96.0437**	**0.3258**

We have obtained information about the relative importance of quality characteristics through a questionnaire for software engineering experts in CE domain. The questionnaire consists of 15 questions which try to find the relative importance between pair of quality characteristics based on reusability. And each question takes an answer within 1, 2 and 3 according to the relative importance. '3' means extreme importance, and '1' means equal importance. We have collected the answers, and then calculated geometric average for each question. Table 1–(a) is a comparison matrix based on the average values. The value of Consistency Index (CI) and Consistency Rate (CR) for comparison matrix is 0.023 and 0.019. CR is less than 0.1, so we can say that the comparison data has consistency. Weights for the quality characteristics, which are derived from AHP are shown in Table 1-(b). Based on the weight, we found that three characteristics – portability, maintainability and reliability are relatively important for CE product software. They cover about 74% of whole quality characteristics. On the other hand, the weight of usability, the fourth characteristic is less than half of the third characteristic. Those lower priority characteristics do not impact software quality much, so we decided to ignore them. We have identified three quality characteristics with high weight as critical quality factors.

3.3. Identifying Metrics for Critical Quality Factors

Critical quality factors and their sub-factors cannot be measured directly, because they are abstract concepts. They can be measured by metrics composing the bottom of the quality hierarchy. Therefore, it is very important to select good metrics describing critical quality factors and their sub-factors.

One more criterion should be considered to identify metrics. Generally, the quality model is utilized for quality evaluation, but it also needs to be utilized for quality improvement. Therefore, we have to identify metrics, which are good for specifying critical quality factors and finding out improvement criteria. The examples of metrics are listed on the first column of Table 2.

Additionally, how to measure metrics is equally important as identifying metrics. Especially, our metrics are based on code analysis, so it is not easy to measure manually. One of easy ways to get metrics for code quality and software structure is to analyze source code with automated static analysis tools. We have enumerated metrics which can be measured by static analysis tools. Then, we discussed with software engineering experts in CE domain to identify metrics, which are related to critical quality factors. Relationship between each metric and critical quality factor is displayed in the second column of Table 2.

Table 2. Mapping Metrics to Critical Quality Factors

METRICS+	R1	R2	R3	M1	M2	M3	M4	P1	P2
# of functions				*					
# of files				*					
Average function call depth				*					
Function call depth					*				
Average cyclomatic complexity					*				
cyclomatic complexity					*				
Maintainability index					*	*			
Lines of physical source				*					
Lines of comment				*					
Ratio of comment lines				*					
# of unused functions				*			*		
# of functions called				*	*				
# of functions calling						*			
# of unused function parameters				*	*		*		
# of global variables					*	*			
# of references to global variables						*			
# of reading from global variables						*			
# of writing to global variables						*			
# of read-only global variables						*			
# of static candidates						*			
# of not referred global variables				*			*		
Packages dependency matrix					*	*		*	*
Files dependency matrix					*	*		*	*
Global variables dependency matrix						*		*	*
# of potential defects	*	*	*						
Potential defects	*	*	*						
Defect distribution	*	*	*						

R1:Maturity	M3:Stability
R2:Fault Tolerance	M4:Testability
R3:Recoverability	P1:Adaptability
M1:Analysability	P2: Replaceability
M2: Changeability	

4. Evaluation of Quality Model

In this section, we describe an evaluation that we have applied our approach to a software module. The software module has been developed for a previous product. Because functions provided by the software module were needed for the other products, developers wanted to improve quality of the software module.

4.1. Quality Evaluation

The first stage for the software quality improvement is quality evaluation. Quality evaluation is useful for understanding the current status of the software and identifying improvement areas. Generally, measurement depends on the experience of an analyzer, and it needs a long time. We utilized two static code analysis tools to avoid such inefficient work.

4.2. Static Analysis Tools

Two static code analysis tools – Understand for C++ and Klocwork has been selected for the quality evaluation. Brief descriptions for them are follows:

- Understand for C++ [7] is a static code analysis tool. It provides metrics and views, which are useful at code reuse such as function complexity, function call depth, usage of global variables and unused functions.
- Klocwork [8] is a static code analysis tool. It has an advantage over detection of potential defects and architecture analysis. Even though Understand for C++ provides many metrics and view, but that's not enough. Therefore, we have detected some potential defects using it.

4.3. Result of Quality Evaluation

We have tried to measure metrics from source code of target software with 'Understand for C++' and 'Klocwork', and then, we found some problems which affect the software quality, as specified in Table 3. We expected that we could improve the quality of software, by solving the issues, because they influence critical quality factors directly. It was a discussed about the issues and their solutions, with developers.

4.4. Conducting Quality Improvement

Table 3. Issues and Solutions which has been found through the Quality Evaluation

Critical Quality Factors	Issues	Solutions
Reliability	Potential defects found	Remove defects if they can be realized.
Maintainability	Too high Average Call Depth	Refactor functions with high call depth exceeding 20.
	Too high Average Complexity	Refactor functions with high complexity exceeding 20.
	Unused Function found	Remove them if they will not be used anymore.
	Read-only global variables found	Replace them with MACRO.
	Global variables which are defined and referred in same file only found	Convert them to Static.
	Unused Variables found	Remove all of them.
Portability	back-calls exist	Relocate functions to abstraction component.

It is recommended to eliminate all issues found by quality evaluation, because they influence software quality directly. But, it is not always possible to eliminate all of them. For example, cost and effort for improvement can exceed benefit. In this case, to leave the issue untreated can be better choice. In other case, some issue is too difficult to improve, so developer may not want to improve it. Sometimes, software metrics designate a bad code block, but it doesn't actually affect the quality of software. The most important thing for quality improvement is to reach consensus for improvement scope and solutions with developers.

It has been decided solutions and scope of the improvement through a discussion. Then the developers focused on the improvement their source code for 2 months. Some improvement points, which were too difficult to improve were rejected.

After improving, we had evaluated the source code again, to see the improvement was successful. See Table 4 for improvement result.

Table 4. Quality Improvement Result

Metrics	Improvement Rate
Number of potential defects	20%
Average cyclomatic complexity	6%
Number of unused functions	100%
Number of read-only global variables	100%
Number of static candidates	100%
Number of not referred global variables	100%
Number of back-calls (PDM)	98%

5. Conclusion and Future Works

It is not easy to evaluate or improve quality of software, because software quality is a complicated concept which consists of attributes in various aspects.

We have identified quality goal which should be achieved by CE product software to define software quality. Quality goal depends on characteristics of domain. Therefore, ensuring reusability has been identified as quality goal of CE product software in this paper. Then, we have specified three critical quality factors – portability, maintainability and reliability from quality characteristics in ISO 9126, based on the relationship with reusability as quality goal. critical quality factors cannot be measured directly, so they should be mapped with measurable metrics. Also, the metrics should be able to specify improvement areas. We have identified metrics which can be measured by static analysis tool for critical quality factors. Finally, we have shown that software quality in CE product can be assured by improvement of metrics, which compose the bottom of our quality model. We expect that our approach can be utilized for not only quality evaluation, but also quality improvement in CE domain.

Even though our quality model is useful for quality evaluation and improvement, it also has some limitations. The metrics composing the bottom of our quality model focus on the source code, even though quality can also be improved though the other way, such as architectural improvement. In the future, our quality model should be extended to include more metrics related to critical quality factors.

6. References

[1] International Organization for Standardization, *ISO9126-1:2001, Software engineering – Product quality*, Part 1: Quality model, Geneva, 2001.

[2] McCall, J. A., Richards, P.K., and Walters, G. F., *Factors in Software Quality*, Vol. 1, 2, and 3, Nat'l Tech. Information Service, Springfield, Va., 1977.

[3] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M., *Characteristics of Software Quality*, North Holland, 1978.

[4] Boehm, B. W., Brown J. R, and Lipow, Mlity, "Quantitative evaluation of software quality", *International Conference on software Engineering*, 1976.

[5] B. Kitchenham and S. L. Pfleeger, "Software Quality: The Elusive Target"*, IEEE Software*, Jan. 1996, pp.12-21

[6] T.L. Saaty, "How to make a decision: The Analytic Hierarchy Process", *European Journal of Operational Research*, Vol 48, pp.9-26, 1990.

[7] S. Toolworks, "Maintenance, Understanding, Metrics and Documentation Tools for Ada, C, C++, Java, and FORTAN", www.scitools.com, 2006

[8] Klocwork Inc., "Improving Software By Reducing Coding Defects",www.klocwork.com, 2004

[9] Balbir Barn, Alan W. Brown, "Methods and Tools for Component Based Development", *Technology of Object-Oriented Languages*, 1998.

[10] Software Engineering Institute, *The Product Line Practice (PLP) Initiative*, Carnegie Mellon University, www.sei.cmu.edu/activities/plp/plp_init.html.

[11] Weiss, david M., Lai, chi Tau Robert, Software Product-line Engineering. Addison-Wesley, 1999