ISSN: 1738-9984

International Journal of Software Engineering and Its Applications

IJSEIA

Vol.8, No.3, March, 2014

SCIENCE & ENGINEERING
RESEARCH SUPPORT SOCIETY

Table of Contents

International Journal of Software Engineering and Its Applications
Vol.8, No.3 (2014), pp.39-46
http://dx.doi.org/10.14257/ijseia.2014.8.3.04

A Case Study of Quality Improvement for Water Resource Management System based on ISO/IEC 9126

Kidu Kim[1] and R. YoungChul Kim[2]
[1]Telecommunications Technology Association
[2]Hongik University
[1]kdkim@tta.or.kr, [2]bob@hongik.ac.kr

Abstract

Software testing organization in the company always tries to verify the completeness of developed code before product release, but difficult to do that. The program, which has shipped without verification of non-functional defects, causes user inconvenience or an irreparable loss. In this paper, we show to improve software quality of water resources management system based on ISO/IEC 9126 including developer-centric functional test.

Keywords: *ISO/IEC 9126, embedded software, SCADA, TAG*

1. Introduction

Most software companies had lack of testing capability with empirical practice of software testing experiences in TTA(Telecommunications Technology Association). Especially, there are lots of empirical practice for development, not for testing and management in the area of software system. Because a product is released only after verifying source code, it has latent faults which causes the end user inconvenience or an irreparable loss.

The TEmb [1] method is published in the Testing Embedded Software by Bart Broekman and Edwin Notenboom. TEmb is a method that helps to assemble a suitable test approach for a particular embedded system. It provides a mechanism for assembling a suitably dedicated test approach from the generic elements applicable to any test project and a set of specific measures relevant to the observed system characteristics of the embedded system.

2. Related Works

2.1. ISO/IEC 9126

ISO/IEC 9126 (1991): Software product evaluation - Quality characteristics and guidelines for their use, which was developed to support these needs, defined six quality characteristics and described a software product evaluation process model. As quality characteristics and associated metrics can be useful not only for evaluating a software product but also for defining quality requirements and other usage, ISO/IEC 9126 (1991) has been replaced by two related multipart standards: ISO/IEC 9126 (Software product quality)[2] and ISO/IEC 14598 (Software product evaluation)[3].

The software product quality characteristics defined in this part of ISO/IEC 9126 can be used to specify both functional and non-functional customer and user requirements. ISO/IEC started work on SQuaRE (Software product Quality Requirements and

Evaluation), a more extensive series of standards to replace ISO/IEC 9126, with numbers of the form ISO/IEC 250mn. For instance, ISO/IEC 25000[4] was issued in 2005, and ISO/IEC 25010[5], which supersedes ISO/IEC 9126-1, was issued in March 2011. ISO 25010 has eight product quality characteristics (in contrast to ISO 9126's six), and 31 sub characteristics.

ISO/IEC 9126 defines terms for the software quality characteristics and how these characteristics are decomposed into sub-characteristics (Figure 1). The sub-characteristics can be measured by internal or external metrics. Software quality can be evaluated by measuring internal attributes (typically static measures of intermediate products), or by measuring external attributes (typically by measuring the behavior of the code when executed).

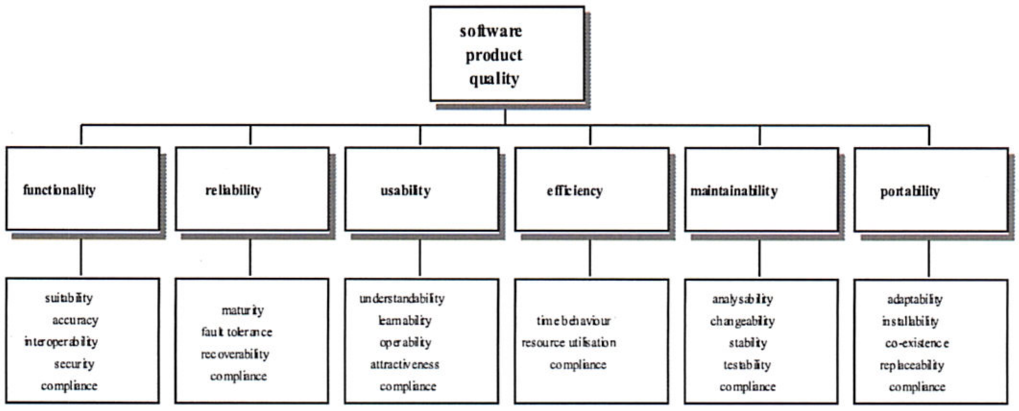
software product quality
functionality
reliability
usability
efficiency
maintainability
portability
suitability accuracy interoperability security compliance
maturity fault tolerance recoverability compliance
understandability learnability operability attractiveness compliance
time behaviour resource utilisation compliance
analysability changeability stability testability compliance
adaptability installability co-existence replaceability compliance

Figure 1. ISO/IEC 9126 Software Product Quality [2]

1) Functionality: the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
2) Reliability: the capability of the software product to maintain a specified level of performance when used under specified conditions
3) Usability: the capability of the software product to be understood, learned, used and liked by the user, when used under specified conditions
4) Efficiency: the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
5) Maintainability: the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
6) Portability: the capability of software product to be transferred from one environment to another.

2.2. Embedded Software

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints such as cellular phone, railway signaling system, hearing aid, missile tracking system. An embedded software is sometimes used interchangeably with firmware, although firmware can also be applied to ROM-based code on a computer, on top of which the OS runs, whereas embedded software is typically the only software on the device in question. Figure 2 shows a generic layout, which is applicable to virtually all embedded systems.

Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Embedded systems usually consist of CPU, memory, input devices, output devices, communication interface, *etc*, and embedded software is stored in non-volatile memory (NVM).

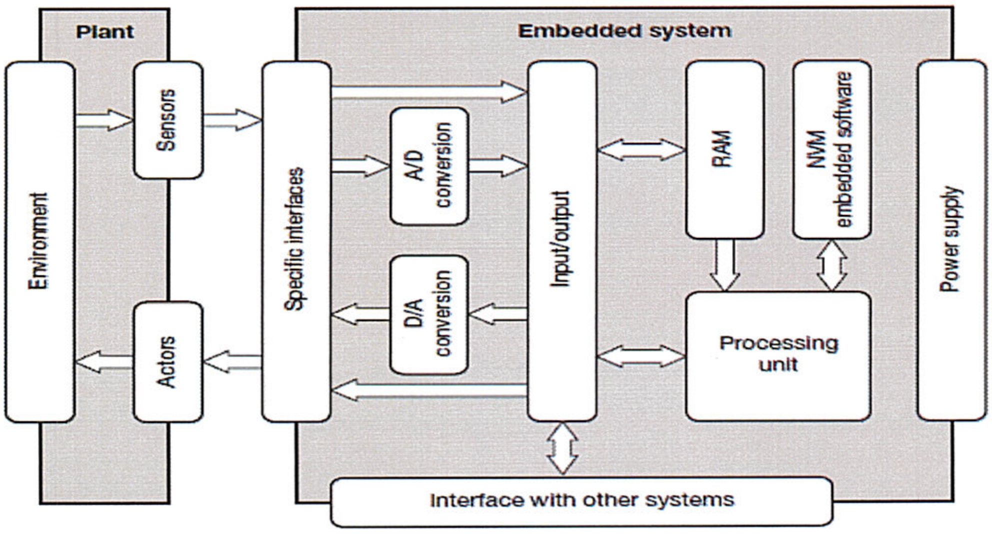

Figure 2. Generic Scheme of an Embedded System [1]

2.3. Embedded Software Characteristics

Each embedded software is developed for a specific purpose, but it has some characteristics as follows:

1) Specific development purpose: Embedded software is typically specialized for the particular hardware, such as refrigerator, car, and cellular phone. It tightly interfaces with hardware.

2) Real-time processing: General-purpose systems typically use most resources effectively, but embedded software has time and memory constraints. Embedded software comes in a wide variety of operating systems, typically a real-time operating system (soft real-time, hard real-time).

3) Mass production: We find easily embedded systems in living: Washing machine, refrigerator, game console. Embedded systems are mass-produced, benefiting from economies of scale.

4) Durability: Embedded system should be operated in high temperature, and humidity even with external impact or some mal-function.

3. Testing Embedded Software (TEmb)

Embedded software testing shares much in common with application software testing. Developer codes a software, compiles, and tests to check the function. This is a developer centric test and is not sufficient to reduce latent faults.

There are just few embedded software testing methodologies. TEmb uses the four cornerstones of structured testing as defined by the test management approach TMap[1]:

1) Lifecycle: This defines which activities have to be performed and in what order. It gives testers and managers the desired control over the process.
2) Infrastructure: This defines what is needed in the test environment to make it possible to perform the planned activities.
3) Techniques: This helps with how to do things, by defining standardized ways to perform certain activities
4) Organization: This defines the roles and required expertise of those who must perform the planned activities and the way they interact with the other disciplines.

However, TEmb testing methodology is for the procedures and management to test effectively, and performed by developers along with product development. It is difficult to apply after the development is completed, and to analyze the fault or test coverage information.

In order to complement this, we use ISO / IEC 9126 software quality attributes and metrics for performing test and managing faults. We describe the case study in the next chapter.

4. Testing Embedded Software for Water Resource Management System

The sequence for testing is described in Table 1.

Table 1. Testing Procedure

Task	Procedure
Preparation	Test planning Setup testing environment
Analysis & Design	Identify test item Write test scenario & test cases
Testing	Testing(make test incident report) Patch the product(fix fault)
Regression	Regression testing Analyze regression result
Reporting	Write test report

4.1. Preparation

Product under test is a water resource management system, which controls water level of a dam, and monitors condition. It requires real-time commands and controls, but we use both simulated system and real system for testing.

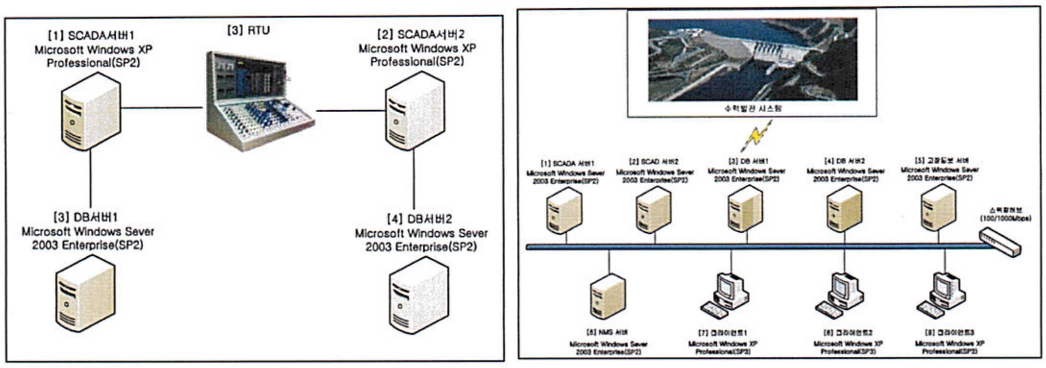

Figure 3. Testing Environment (Simulated System and Real System)

There are SCADA(Supervisory Control And Data Acquisition) server and DBMS(Database Management System) server in each testing environment. In simulated system, there is a RTU(Real Time Unit) to generate virtual data and execute control.

4.2. Analysis and Design

With consideration of the characteristics of embedded software, we wrote test cases, and added more test cases in progress of testing. The test case has a unique ID, expected result, and execution result.

Table 2. Test Case

TC ID	S2-01	TC name	Alarm set
Prerequisite	System running	Test Result (P/F)	P
No.	Test procedure	Expected result	Execution result
1	Select ‘alarm’ tab in 'Analog Input Tag'	move to 'alarm'	move to 'alarm'
2	Input integer(1) to 'High'	1	1
3	Input integer(10) to 'High'	10	10
4	Click 'Apply' button	Execute Alarm	Sound an alarm

Table 2 is a part of test cases for ‘Alarm set’. When writing test cases, we validated test cases through review meeting.

4.3. Testing

When performing the test, we found several faults, and classified based on ISO / IEC 9126 quality characteristics. We found 90 faults on 4 quality characteristics (functionality, reliability, usability, portability) (Figure 4).

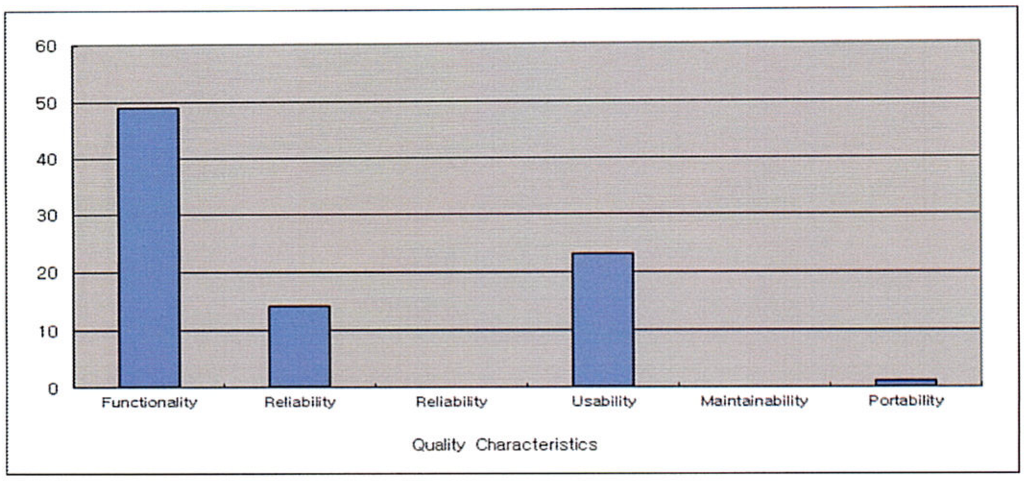

Figure 4. Faults per Quality Characteristics

4.4. Regression Testing

After correcting the fault, we performed regression testing, and validated the system. We also assert the system had no defect with 7 days (over 168 hours) continuous running.

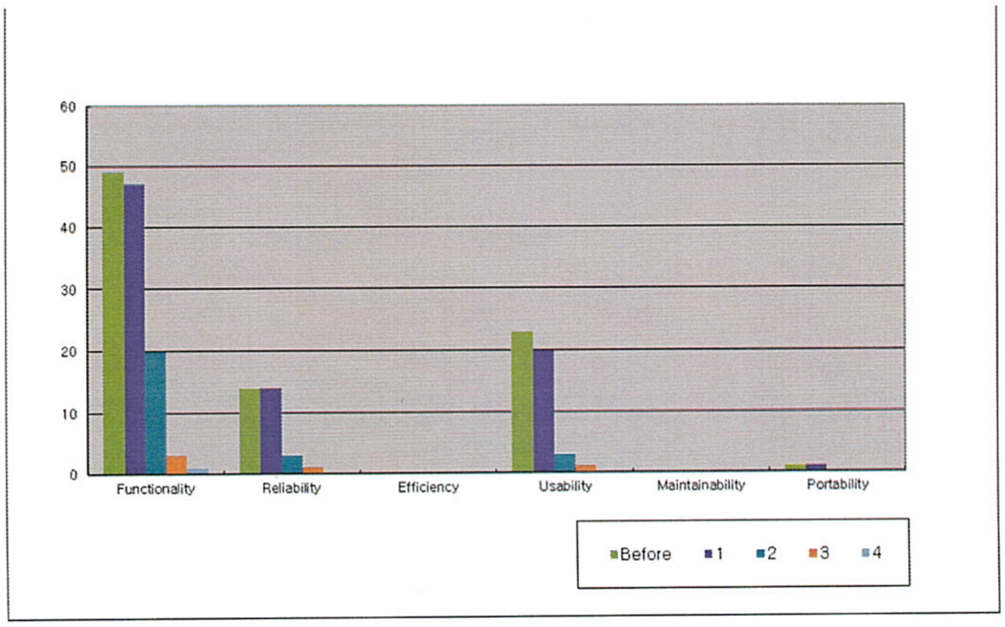

Figure 5. Faults per Quality Characteristics after Patch

There was 4 patches to fix all found faults. In 1st stage, developer fixed faults without information of quality characteristics. After 2nd stage, developer fixed faults with information of quality characteristics. Rate of patch was only 6% in 1st stage, but 68% in 2nd stage. (Table 3). So providing information of quality characteristics helped developer to comprehend what was error. We also found latent fault, during patch.

Table 3. Rate of Patch

Content	1st stage	2nd stage	3rd stage	4rd stage
Rate of Patch	6 %	68 %	80 %	100 %

5. Conclusion

Previous embedded software testing was verifying source code using functional testing, so testing was not sufficient except program function. In this paper, we tested an embedded system for water resources management based on ISO / IEC 9126. As a result, providing information of quality characteristics improved efficiency of modification 60%, and prevented mistake of developer's modification. However, the case study of ISO/IEC9126 based test is not sufficient. Later, we will derive items to improve the quality except the quality characteristics of ISO / IEC 9126.

Acknowledgements

This work was supported by the IT R&D Program of MKE/KEIT [10035708, "The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"] and Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (2013R1A1A2011601).

References

[1] B. Broekman and E. Notenboom, "Testing Embedded Sofwtare", **(2002)**.
[2] ISO/IEC TR 9126, Software engineering-Product quality-Part 1, 2, 3, 4, **(2005)**.
[3] ISO/IEC 14598, Information Technology-Software Product Evaluation-Part 1,2,3,4,5,6, **(2005)**.
[4] ISO/IEC 25000, Software Engineering-Software Product Quality Requirements and Evaluation-Guide to SQuaRE, **(2005)**.
[5] ISO/IEC 25010, Software Engineering-Software Product Quality Requirements and Evaluation-Quality Model, **(2005)**.

Authors

Kidu Kim, received the B.S. and M.S. degree in Software Engineering from Hongik University Graduation, Korea in 2005. He is currently a researcher in Telecommunications Technology Association (TTA). His research interests are in the areas of Test Maturity Model (TMM) and Test Process.

R. Young Chul Kim, received the B.S. degree in Computer Science from Hongik University, Korea in 1985, and the Ph.D. degree in Software Engineering from the department of Computer Science, Illinois Institute of Technology (IIT), USA in 2000. He is currently a professor in Hongik University. His research interests are in the areas of Test Maturity Model, Embedded Software Development Methodology, Model Based Testing, Metamodel, Business Process Model and User Behavior Analysis Methodology.