Coverage analysis method using quality characteristics

Daiju Kato
Nihon Knowledge Co., Ltd.
Tokyo, Japan
d-kato@know-net.co.jp

Hiroshi Ishikawa
Tokyo Metropolitan University
Tokyo, Japan
ishikawa-hiroshi@tmu.ac.jp

***Abstract*— Requirement of Must-Be Quality for software has changed over time, and the awareness of software quality has become increasingly strong. Therefore, an objective and easy-to-understand method for analyzing software quality is necessary. By utilizing SQuaRE (ISO/IEC 25000 series) in software development projects, it is possible to proceed with software development with comprehensive quality in mind. However, in order to utilize these quality characteristics, the project is needed to have traceability of quality in various phases or sprints. Therefore, analyzing the quality using by various evidence under development project provides an effective means of explanation of the quality logically and enables us to judge the quality under standard rules. In this paper, we propose a quality coverage analysis method using quality characteristics for software within a general software lifecycle.** *(Abstract)*

Keywords- quality analysis, software engineer, quality characteristics, SQuaRE (key words)

I. INTRODUCTION

The Kano Model[1] is a type of customer satisfaction model and the model uses data obtained from questionnaires to rank product quality (function and performance) in terms of attractiveness (differentiation) and affordability (indispensability), etc., to visualize the characteristics of products from the customer's point of view and stimulate discussion in design and development.

In the world of software, I feel that this Kano model of quality has been broadly interpreted and used in the same way as minimum requirement of quality. However, we believe that this quality requirement is changing with the times like below bullet. The quality characteristics in parentheses at the end indicates those that match the product quality.

- Quality that was commonplace 10 years ago

-Functions work correctly as required (Functional Suitability)
-No response problems (Performance Efficiency)

- Quality that is commonplace today

-Safe to use (Security)

- Quality as a matter of course in the future

-Quality with the user in mind (Effectiveness - Quality in Use-)
-Safe and secure to use (Freedom for risk – Quality in Use-)

*Quality characteristics in parentheses

Ten years ago, Must-Be Quality indicated that a function worked exactly as required and that there were no performance issues. However, this alone is worrisome in a world where a variety of devices are connected to the Internet. Every week, numerous cyber incidents are distributed by the CERT Coordination Center[2] , and Microsoft delivers security updates on the second Tuesday of every month[3]. In addition, antivirus patterns are being delivered by anti-virus software vendors quickly. The ability to use software with peace of mind, i.e., to use software without being aware of vulnerabilities and security as well as features and performance, has become a minium requirementment of quality these days.

ISO/IEC 25010[4] defines two quality models. A system must meet the explicit and implicit needs of the various people who use the system, called stakeholders, and satisfying them is considered to be quality. This quality is categorized by characteristics, and the standard defines two models: product quality and Quality in Use. There are other data quality models defined in ISO/IEC 25012[5], which defines the quality of the data used in the system.

The quality model consists of characteristics and sub-characteristics that make up the characteristics. The product quality model classifies the quality of a software or system product into eight characteristics. This product quality provides the 'quality of things'. It means the quality of a thing because it is a quality that the product itself has.

On the other hand, quality in use is the quality that people receive or feel when they use the product. It is called as "user quality" and it is a quality that is perceived mainly by the user. Quality in use is closely related to product quality, and quality in use will not improve unless product quality is exhaustive.

SQuaRE consists of five divisions and defines the means to achieve quality-conscious software development, Figure.1. Utilizing SQuaRE in your development projects will enable you to develop software with quality characteristics in mind.

However, to make full use of quality characteristics, SQuaRE can be used from the quality requirement phase of a development project, or all test work must have test cases or

test criteria that are mapped to quality characteristics to classify the ensured quality into quality characteristics. Since quality characteristics provides essentially a classification of quality, they are effective in measuring the coverage of the quality of the final product. Therefore, we developed a method to achieve coverage analysis of quality with classification of the quality of software during a development project with ISO/IEC/IEEE 12207 [6] and ISO/IEC/IEEE 15288[7]. Those standards define software life cycle model and classification of activity at development process with mapping of product quality characteristics.

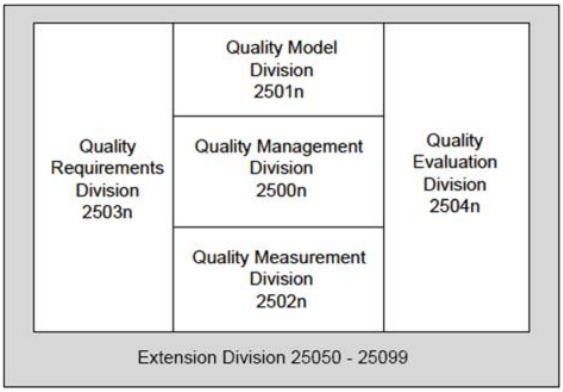

Figure 1. Organization of SQuaRE series

II. CLASSIFICATION OF ACTIVITIES BY QUALITY CHARACTERISTICS

This section proceeds with the quality coverage analysis from the following artifacts created in a typical software development project.

- Project plan and test plan
- Various documents: requirement spec, design spec, test cases and test procedures
- Bug information during the project
- Review results and Test analysis report
- Software and test data

The project plan generally describes goal of the quality requirements. Also, the document indicates how to fulfill the quality requirement even though the project use waterfall development process or agile process. The project plan includes test process to list the test types to be conducted and will contain the idea of the criteria for each test type.

Since the quality requirements written in the project plan are the quality goals of the final deliverable software, it is possible to define the required quality by classifying the quality requirements by quality characteristics. It is necessary to analyze both the process and the deliverables to see if the developed software has the quality that is the goal.

The first step to find out if the process was a quality-building process is to find out which quality characteristics the various activities during the project have an effect on. Although it is very time-consuming to do this task from scratch, ISO/IEC 30130[6] summarizes which quality attributes are mapped to various activities, TABLE 3. For example, 'Test Design' described in the first part of this table is work for all quality characteristics and the second 'Risk Base Priority' is work for functional suitability, reliability and performance efficiency.

Table 1 shows the results of mapping common test types to quality characteristics. You can refer to this table to map the quality characteristics of the test types written in the test plan.

TABLE I. MAPPING OF QUALITY CHARACTERISTICS TO TEST TYPES

Test types	Quality Characteristics
Accessibility Testing	Usability
Compatibility Testing	Compatibility
Conversion Testing	Functional Suitability
Disaster Recovery Testing	Reliability
Functional Testing	Functional Suitability
Instalation Testing	Portability
Interoperability Testing	Compatibility
Localization Testing	Functional Suitability
	Usability
	Portability
Maintainability Testing	Maintainability
Performance-Related Testing	Performance efficiency
Portability Testing	Portability
Reliability Testing	Reliability
Security Testing	Security
Usability Testing	Usability
Stress/Load Testing	Performance efficiency
	Reliability
Screen Transition Testing	Functional Suitability
	Usability

From the above approach, it will be possible to summarize which quality characteristics the development project is performing for from the activities listed in the project plan or sprint plan and the test plan.

III. MAPPING QUALITY CHARACTERISTICS TO REQUIREMENTS

The next step is to map the quality requirements to quality characteristics to identify the quality that the final product, the software, will require. If the quality requirements are not clear, it is a good idea to categorize them into functional requirements, performance and load requirements, user interface requirements, etc., and map each requirement to a quality characteristic. If there is a service specification, the listed items can also be classified as quality characteristics.

Maintainability requirements are generally not included in requirement definitions. Of course, You can refer to the project plan or project rules to determine how software maintenance will be performed. The quality characteristic probably contains coding style rules for programming or testing way for using some stab or debugging technique.

Ideally, the requirements derived from each requirement should also be classified as a quality characteristic. Typically, requirement items are mapped to a single quality characteristic so that the requirements required by the final product, the software, can be counted for each quality characteristic. Since the number of man-hours of classification work increases proportionally with the size of the software, requirements can be sampled or, at worst, only requirements can be classified into quality characteristics.

This work completes the classification of quality characteristics of the quality required for the software.

IV. Mapping Various Evidence to Quality Characteristics

Then, classify all the review evidence over the life of the project, e.g., design reviews, code reviews, test design reviews, etc., into quality attributes. Rather than simply mapping activities to quality attributes, we classify criteria into quality attributes. For example, in the case of API design, the criteria and review reports show whether the reviewers check not only the implemented functionality itself, but also error sequencing and recovery methods, performance and load considerations, authentication methods, and vulnerability responses and so on.

The findings that are identified and corrected during the review are also classified by quality characteristics. Corrected bug information is important evidence to ensure the quality of the corresponding quality characteristic.

Finally, we refer to the criteria for each test type performed to see if the quality characteristics mapped to the test type are ensured by having met the criteria. For example, suppose the criteria are set for performance testing, Table 2. If the criteria are too vague to map the quality characteristics of the criteria, the test cases are checked and classified. In addition, bugs found and fixed in the test as well as in the review report are classified according to each quality characteristic to check whether the quality characteristics are maintained or not.

TABLE II. Mapping Quality Characteristics to Test Criteria at Stress Testing

No	Test	Criteria	Quality Characteristics
1	Single operation	The CPU load should return to normal after each process.	Performance efficiency
2	Single operation	Memory is released after each process	Performance efficiency
3	Use-case	If a new process occurs while multiple processes are in progress, it should not result in an error.	Reliability
4	Use-case	The CPU load becomes 100% and other processes are not interrupted while multiple processes are running.	Reliability
5	Peak operation	The process is carried out even if the load is twice the expected workload.	Performance efficiency Reliability

If there is a description of quality in the test report, we consider whether the description or results can be mapped to some quality characteristics as well.

Categorizing the evidence of these activities into quality characteristics will clarify the comprehensiveness of the quality of the final product.

Figure 2 shows the way of mixed up classify for coverage analysis method using quality characteristics. PQ in the figure stands for Product Quality.

V. Creating a Coverage Analysis of Quality

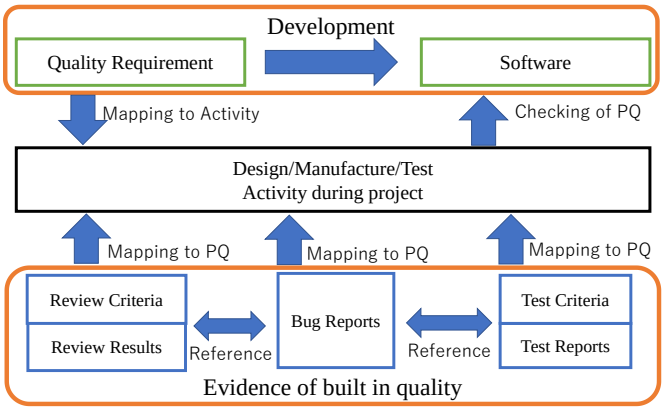

Figure 2. Coverage analysis method using quality characteristics

Using Quality Characteristics

Once all the classification work for completeness analysis using quality characteristics is completed, the results of categorizing the quality requirements are compared with the results of categorizing the final deliverable, the software, to ensure that the quality is on target. The broad analysis is determined by using a radar chart as like Figure 3 to visualize the differences between requirement and results.

The areas that have a large difference between requirements and results indicate possibility of the unclear qualities, which can be pursued by checking the specific classified results and investigating the causes.

In general, functional conformance requirements are often fulfilled. But reliability is not ensured if the scope of impact of a bug fix is not adequately checked or if the functional conformance test is not re-run even though effect of fixed some bug found in a test related to performance efficiency has an impact on functionality. It is also a good idea to analyze whether their impact on relevant quality characteristics has been verified or not, when the bugs which are classified as non-functional conformance have been fixed.

If there is a large difference between the two, you can run some of the tests and check the results of the analysis with your team members to improve the validity of the results.

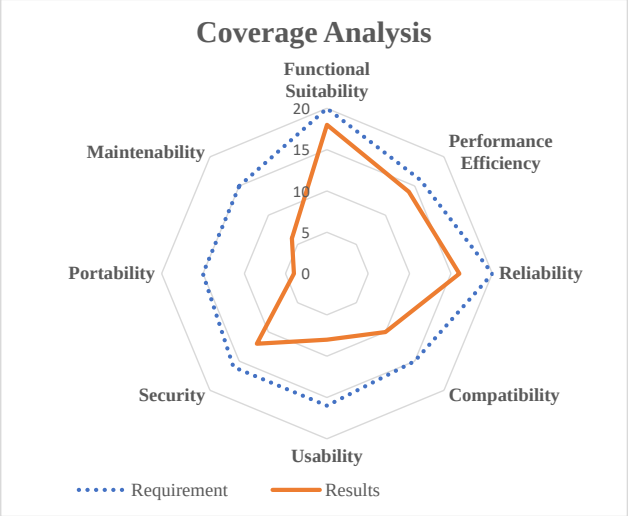

Figure 3. Coverage analysis by radar chart

VI. CONCLUSION

Quality characteristics enable us to objectively and logically classify the quality of software product, and conduct an exhaustive analysis of whether the quality requirements are being met or not. Although it is most desirable to use SQuaRE from the beginning of a development project, quality analysis using quality characteristics can be performed even after the project is over by using the method described here. In addition, it is possible to identify the quality that is considered to be unsecured from the analysis results, and therefore, it is also possible to consider policies for strengthening weaknesses.

Thus, quality comprehensiveness analysis using quality characteristics not only visualizes the quality of software, but it is also effective as a means to improve quality.

It is also possible to classify test techniques, process transfer criteria and test reports in sprints by quality characteristics. In the case of derived development, it is possible to judge the appropriateness of quality quantitatively for each quality characteristic by using quality data from past development and quality characteristic quantities defined in ISO/IEC 25022. We intend to further improve the quality comprehensiveness analysis method using these quality characteristics so that it can be used as a quality monitoring method that enables objective visualization of quality.

REFERENCES

[1] Kano,N., Seraku, N., Takahashi, F., Tsuji, S. : Attractive Quality and Must-Be Quality; Quality Vol.14(2), pp147-156, JSQC, 1984, (Japanese)

[2] CERT Coordination Center, https://www.sei.cmu.edu/about/divisions/cert/index.cfm

[3] Microsoft Security Update Guide, https://portal.msrc.microsoft.com/en-us/security-guidance

[4] ISO/IEC 25010: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models (2011)

[5] ISO/IEC 25012: Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Data quality model(2008)

[6] ISO/IEC/IEEE 12207: Systems and software engineering — Software life cycle processes(2017)

[7] ISO/IEC/IEEE 15288: Systems and software engineering — System life cycle processes(2015)

[8] Kato, D., Okuyama, A., Ishikawa, H. : Introduction of test management based on quality characteristics, IWESQ 2019, 2019.

[9] Kato,D., Ishikawa, H. : Develop Quality Characteristics based quaity evaluation process for ready to use software products, JSE-2016, February, 2016.

[10] Kato,D., Okuyama, A., Ishikawa, H.: Use proactive evaluation process for 'Quality in Use', Seventh World Congress for Software Quality, 2017.

TABLE III. SUMMARY OF CAPABILITIES WITH CHARACTERISTICS IN ISO/IEC 30130

Capabilities	Categories												Characteristics										
	Dynamic test execution			Test environment	Code analysis		Test management						software quality characteristics								Granularity		
	Input for dynamic test execution	Dynamic test execution	Test data repository	Test environment	Input for code analysis	Code analysis	Test plan	Test asset	Quality record report	Test completion report	Ver if ication and validation	Test status report	Functional suitability	Reliability	Usability	Performance efficiency	Maintain ability	Portability	Compatib ility	Security	Smallest unit	Intermedia te units	Largest unit
Test design	○												○	○	○	○	○	○	○	○	○	○	○
Risk Based Priority	○												○	○		○				○	○	○	○
Test execution controll, Automated test execution		○											○	○	○	○	○	○	○	○	○	○	○
Caputure,Playback		○											○	○	○	○	○	○	○	○	○	○	○
Keyword driven test case		○											○									○	○
Test comparator		○											○										○
Debugging		○											○								○	○	○
Dynamic analysis		○											○			○					○	○	○
Monitoring		○											○	○		○					○	○	○
Coverage measurement		○											○								○		
Security testing		○																		○			○
Test data preparation, Test data generation			○										○	○	○	○	○	○	○	○	○	○	○
Stress testing,Load testing			○											○		○							○
Performance testing			○													○					○	○	○
Data validation and verification			○										○			○				○		○	○
Database validation and Verification			○										○			○				○		○	○
Emulators,Simulators				○									○			○					○	○	○
Unit test framework				○									○								○		
Automated environment set up				○									○	○	○	○	○	○	○	○	○	○	○
Runtime envrionment management				○									○	○	○	○	○	○	○	○		○	○
Review					○								○	○	○	○	○	○	○	○	○	○	○
Code analyzer						○								○			○	○		○	○		
Codebased security testing						○																	
Test management							○			○			○	○	○	○	○	○	○	○	○	○	○
Test Asset configuration management								○					○	○	○	○	○	○	○	○	○	○	○
Incident management									○				○	○	○	○	○	○	○	○	○	○	○
Defect management, Defect tracking,Bug tracking									○				○	○	○	○	○	○	○	○	○	○	○
Test monitoring												○	○	○	○	○	○	○	○	○	○	○	○
Relating of data that serves as the basis for Verification and Validation Reports											○		○	○	○	○	○	○	○	○	○	○	○
Verification and Validation Report											○		○	○	○	○	○	○	○	○	○	○	○