Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

A Method of Software Requirements Analysis Considering the Requirements Volatility from the Risk Management Point of View

Yunarso Anang [1,2], Masakazu Takahashi [1], and Yoshimichi Watanabe [1]

[1] University of Yamanashi, Department of Computer Science and Engineering, 4-3-11 Takeda, Kofu-shi, 400-8511, Japan
[2] Institute of Statistics, Department of Computational Statistics, Jl. Otto Iskandardinata No. 64C, Jakarta Timur, 13330, Indonesia
{g14dma01,mtakahashi,nabe}@yamanashi.ac.jp, anang@stis.ac.id

Abstract. To accelerate the development life cycle of a software product, the incremental development life cycle models such as spiral and agile model have been introduced. However, due to the immaturity of the specification during the incremental cycle, the number of changes of requirements is big. Even with conventional waterfall model, there is no way to avoid the change of requirements to occur after the phase of requirements. So, rather than try to perform requirements analysis to obtain perfect coverage of requirements, it is easier to just accept the potential of requirements change as a risk. In this paper, we describe our study about a method of software requirements analysis while considering the requirements volatility as a risk. We use Quality Function Deployment as the base method of requirements analysis, while we apply R-Map as a tool for risk assessment. By using the method, we can have a better understanding of requirements volatility from the risk management point of view. We use actual software changes tracking record to obtain the risk of changing, and we evaluate our proposed method by applying the method to a real software product as our case study.

Keywords. software requirements volatility, risk management, quality function deployment, r-map

1 Introduction

In today's software development, the customer is asking for more rapid delivery of the software product, whether it is fully completed, partly completed, or even without any working functionality except for the visual user interface. The customer demands the ability to look at the software product at a very early time during the development life cycle in order to ensure customer requirements are fulfilled in the product. To accelerate the development life cycle of a software product, the incremental development life cycle models such as spiral and agile models have been introduced. However, it is widely acknowledged that requirements volatility is inevitable and has a great impact on the software development life cycle [1]. Due to the immaturity of the software specification during the life cycle of these models, customer requirements may be changed and the specification should be modified to accommodate those changes. Customer requirements are getting mature, and the software specification is getting mature too, but the internal design and its implementation may need a big modification in order to accommodate those incremental changes. The consequences are as follows. The delivery time may become longer, the quality of the software product may be compromised, and the overall cost may grow. Nevertheless, those changes of requirements are inevitable during the development life cycle. Thus, rather than try to avoid those changes by obtaining a perfect specification of requirements, it is easier to just accept this potential for changes as a risk and find a way to anticipate those changes. This research focuses on how to anticipate the changes occurred during the development life cycle so that the implementation of the changes has minimal impact on the already designed or even to the already running software product.

This paper proposes a method to analyze the potential for changes of software requirements considering the risk of changing. The potential for changes is also known as volatility. Hereinafter, we will call the values representing the potential for changes as the degree of volatility. This method assumes that the software requirements are elicited and analyzed using Quality Function Deployment (QFD) as a method to clarify the voices of customer, and defines the product quality as well as the business functions of the product based on them, and takes them in the whole development process [2]. In the proposed method, the customer requirements are deployed to the software functions and architectural design elements subsequently. The degree of volatility of architectural design elements is determined based on the empirical data of software changes in the past projects. In this paper, we introduce the risk assessment using R-Map to obtain the risk of changing which is then considered as the degree of volatility. R-Map is a method for risk assessment which has been developed by the Union of Japanese Scientists and Engineers (JUSE) in 1999 [3].

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

Furthermore, we use those values of the degree of volatility of the architectural design elements obtained from R-Map to obtain the degree of volatility of software functions and customer requirements subsequently. Knowing the degree of volatility of the customer requirements before the actual design and implementation phases enables us to anticipate the future changes and take an appropriate action.

In this paper, we describe our proposed method using an illustration diagram and a simple example is provided. In order to obtain the risk of software changing, we have conducted a survey to gather software changes record and use it in software changes risk assessment. Furthermore, in order to evaluate our proposed method, we have conducted a case study to study how our proposed is capable of supporting the analysis of software requirement volatility in early phase of software development.

The rest of this paper is organized as follows. Section 2 describes the related studies. Section 3 describes the proposed method with application to a simple example, and the risk assessment using R-Map with a survey we have conducted. Section 4 describes our case study. Finally, we conclude this paper in Section 5.

2 Related Studies

In this section we describe the studies related to the proposed method. Related studies we have covered consist of studies regarding software requirements volatility, studies regarding software architectural design, studies regarding quality function deployment especially in the domain of software development, and finally studies regarding risk management especially those using R-Map as the method for risk assessment.

Software requirements volatility has been studied for quite a long time. Dev et al. have conducted a comprehensive literature studies regarding requirement volatility during software development process [1]. From the review, it is confirmed that it is widely acknowledged that requirements volatility is inevitable and has a great impact on the software development life cycle. Sharif et al. have conducted an empirical investigation on the impact of changing requirements on software project cost [4]. The result shows the significant relationship of software development life cycle phase and cost that if changes occur in later phases more rework for the implementation is required. This result indicates that the earlier the requirement volatility can be identified, the fewer reworks will be required. Nurmuliani et al. in their analysis of requirements volatility found that change request occurs starting at the design phase [5]. This result indicates that at the design phase where the software architectural design is decided, it is possible to estimate the requirements volatility from the software architectural design elements. Some requirements were not captured during the initial product definition or discovered after detailed design analysis. The paper identifies the root causes of requirements volatility as: (1) changes in market demands which is a reflection of changes in customer needs; (2) developers' increased understanding of product domain during detailed analysis; and (3) organizational considerations which is most likely related to business goals and policy such as functionality enhancements, product strategy, or scope reduction. Lim et al. proposed a method to classify requirements into layers in order to anticipate the volatility of the requirements [6]. However, the method has the lack of metric and there is no distinction between requirements and software function.

Software architectural design is a process to define software architecture, component, module, interface and data for the software system, in the constraint to satisfy the software requirements [7]. In this process, rather than developing from the scratch, most projects are adopting one of the well-known and explicit architectural patterns depending, or not depending, on the domain of the system [8][9]. Software architecture can also be selected by reusing the already trusted components providing functions for a specific domain, business application, user interface, or network and these components provide the framework for the software application [10]. In the proposed method, there is a step of deploying the architectural design elements from the software functions. There, we can choose from the already suggested architectural patterns. In this paper, we adopt the principal 3-layer architecture which consists of three layers: presentation, domain and data source [11]. The presentation layer provides functionalities to display information and capturing user input; the domain layer provides functionalities in the business logic; and the data source layer provides functionalities to communicate with the back-end services. This layering concept helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction [8]. Layers are sorted vertically, from the lowest level of abstraction at the bottom to the highest level of abstraction at the top. The lower layer also has less potential for changes than those above it. In the 3-layer architecture, the presentation layer is on the top with the highest potential for changes and the data source layer is on the bottom with the lowest potential for changes. Anang et al. have proposed a method of applying the layering concept to the software requirements analysis and architectural design [12].

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

The studies in adapting QFD in the domain of software development have also been conducted for quite a long time. Haag et al. stated that the adaptation of QFD in software development is a front-end requirements solicitation technique, adaptable to any software engineering methodology [13]. Hertzwurm et al. have given an overview of the state of the art of QFD in software development introducing several applications of QFD in software development [14]. Gloger et al. described how QFD can be employed in analyzing software architecture [15]. The paper also indicates the effect of changes to the development time regarding the architectural design while did not involve them to the analysis of requirements volatility.

And finally, the studies related to risk management especially those using R-Map as the method for risk assessment. K. Matsumoto from the Product Safety Advisor in Products Safety Technology Center, Japan presented a seminar at the China Quality Association and the Japan Science and Technology League in June 2010. He was addressing the product safety management and the risk plot (R-Map) method. He described how he was using the R-Map to identify the risk of revolving doors of Japan, when the revolving doors of the Roppongi Building Tokyo killed a 6-years old boy on 2004. His address has been translated to English and was brought to our attention [16]. The R-Map Practice Workshop within Union of Japanese Scientists and Engineers (JUSE) also published a free textbook introducing the R-Map [17]. Y. Sakai presented a risk-based validation method focusing on the product line methodology and proposed two approaches to validate the embedded software [18]. In the paper, he described how he use the risk analysis matrix to assess the hazard, cause, level of concern, likelihood/failure rate, and the method of control of the electrical pot. He described that by writing the risk analysis matrix, the safety and reliability of the product can be well managed. Y. Kyoya et al. proposed a method to reduce the future risk in software development using the combination of QFD and Software-FMEA [19]. In the paper, Software-FMEA was used to complement customer requirements, which will be the input of QFD. The result of QFD is then used to set the target scope of each risk in Software-FMEA.

3 Proposed Method of Software Requirements Analysis

In this section, we describe our proposed method of software requirements analysis. The proposed method consists of two tasks, one is for the analysis of software requirements volatility using QFD, and the other is for the assessment of software risk of changing using R-Map. The first task is conducted to apply the QFD concept to the software customer requirements to derive the software functions and software architectural design elements respectively. In this task, the degree of volatility, which is representing the potential of changing, will be assigned to the architectural design element. To obtain the degree of volatility, in this paper, we propose to employ the assessment of software risk of changing using the R-Map. The result of R-Map, the risk of changing, is then considered as the degree of volatility.

3.1 Analysis of Software Requirements Volatility Using QFD

QFD has been studied to be used in software requirements analysis. In this paper, we apply QFD to the early stage of software development, to analyze the customer requirements and to determine the potential of changing, which is commonly known as the requirement volatility. Requirement with a high degree of volatility has the potential of changing in the future. By understanding the requirement volatility, we can be prepared for future changing.

The idea of the proposed method is as follows. First, there is a fact that it is difficult to identify the degree of volatility from the customer requirements themselves until the customer see the software product at the level where the customer can see how their requirements are reflected. Considering that, if we could extract the architectural design elements of the software product, which we could determine the degree of volatility from the architectural design pattern or from the past projects, then we could estimate the degree of volatility of the customer requirements back from the degree of volatility of architectural design elements.

Figure 1 shows the steps of the proposed method as well as the illustration of the deployment tables and the calculation of the values of the degree of volatility as the product of the proposed method. The proposed method consists of 5 major steps.

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

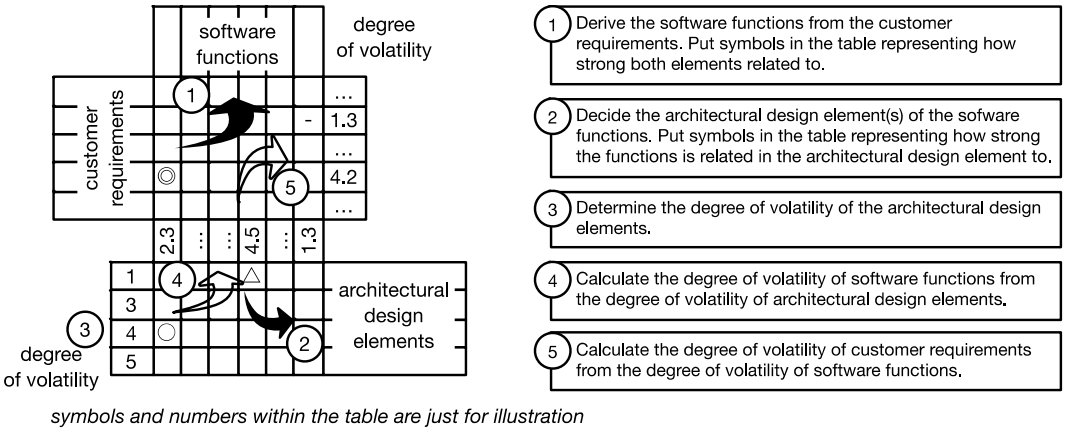

Figure 1. Deployment of Customer Requirements to Obtain the Degree of Volatility

STEP 1: Deploy Software Functions from Customer Requirements.

From the given customer requirements, we derive the software functions which are needed to fulfill the customer requirements. In this step, we also define the relationship between customer requirements and derived software function. The product of this step is a list of software functions and a two-dimensional table representing the relationships between the customer requirements and the software functions. The relationship uses a scale number or symbol. As described in ISO 16355-1:2015 about quantifications of information in QFD, there are different approaches to quantifying relationships [20]. In this paper, we use the classical QFD matrices which use three levels of relationships described as strong, moderate, and weak, with corresponding values of 5, 3, and 1, and symbols of "◎", "○", and "△", respectively. The form of and the procedure to build this two-dimensional table is similar to the typical house of quality in common QFD.

STEP 2: Deploy Architectural Design Elements from Software Functions.

In this step we decide what is the architectural design elements which fit for each software function, as well as the relationship between software functions and the architectural design elements. The product of this step is a list of architectural design elements and a two-dimensional table representing the relationships between software functions and architectural design elements. The relationship uses the same approach as that described in STEP 1.

STEP 3: Determine the Degree of Volatility of Architectural Design Elements.

For each architectural design element, we determine the degree of volatility, based on the architectural design pattern or based on empirical data of software change from the past projects. We use scale numbers to represent the degree of volatility. In this paper, again, we use scale numbers where the lowest number represents the lowest degree of volatility and the highest number represents the highest degree of volatility.

STEP 4: Calculate the Degree of Volatility of Software Functions.

After we have these values of the degree of volatility of each architectural design element, then in this step, we calculate the values of the degree of volatility and their ratios for each software function. The calculation is conducted by using the information about the relationships between software functions and architectural design elements. The procedure of calculation is similar to the procedure of converting the weight of customer requirement, for example, the quality characteristic in common QFD.

STEP 5: Calculate the Degree of Volatility of Customer Requirements.

And in the last step, we calculate the values of the degree of volatility and their ratios for each of customer requirements. The calculation is based on the values of the degree of volatility of the software functions obtained in the

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

Table 1. An Example of Software Requirements Volatility Analysis

(a) Customer Requirements Deployement

Customer requirements		Software functions f1	f2	f3	f4	f5	f6	f7	Degree of Volatility	Degree of Volatility ratio (%)
	r1			◎		△			80	9.5
	r2		◎			△			130	15.4
	r3	○		◎			△		115	13.6
	r4				◎			○	170	20.1
	r5		◎		◎		△		275	32.5
	r6	◎				△		○	75	8.9
	Degree of volatility	5	25	15	25	5	25	15		
	Degree of volatility ratio (%)	4.3	21.7	13.0	21.7	4.3	21.7	13.0		

(b) Software Functions Deployment

Software functions		Architectural design elements a1	a2	a3	Degree of Volatility	Degree of Volatility ratio (%)
	f1	◎			5	4.3
	f2			◎	25	21.7
	f3		◎		15	13.0
	f4			◎	25	21.7
	f5	◎			5	4.3
	f6			◎	25	21.7
	f7		◎		15	13.0
	Degree of volatility	1	3	5		

previous step. The calculation is conducted by using the information about the relationships between customer requirements and software functions. As in the previous step, the procedure of calculation is similar to the procedure of converting the weight of customer requirement, for example, to the quality characteristic in common QFD.

After we obtain the degree of volatility of customer requirements (or the ratio of the degree of volatility), then we can study these values to analyze the customer requirements volatility. If the ratio is low, then the requirement may have a low potential for changes during the development process. We may design the implementation of this requirement using a strict architectural design. But if the ratio is high, then it may be better to design the implementation of this requirement using a loose architectural design because the potential for changes in the future is high. Or if the ratio is high, we can determine that the requirement will need to be classified in more detail specification to anticipate changes in the future.

Table 1 shows an example of the deployment table and the calculation of the degree of volatility of the proposed method. From this example, we can see that the requirement r4 and r5 are having a high ratio of the degree of volatility while the requirement r1 and r6 are having a low ratio of the degree of volatility.

3.2 Assessment of Software Risk of Changing Using R-Map

R-Map is a method for risk assessment which has been developed by the Union of Japanese Scientists and Engineers (JUSE) in 1999. By quantifying the severity of impact and the frequency of occurrence, and showing them in a matrix, a method to visualize the risk has been provided. This will make the level of risk of the targeting product become more clear. Also, this will make the process of clarification of the reduction countermeasure of the risk become easier. Detail information about the method of risk assessment using R-Map is provided in books titled "Risk Assessment Handbook for Consumer Product—First Edition" (in Japanese) and "Risk Assessment Handbook—Practice" (in Japanese), which both have been published by Japan Ministry of Economy, Trade, and Industry (METI) on May 2010 and June 2011 respectively [20][22].

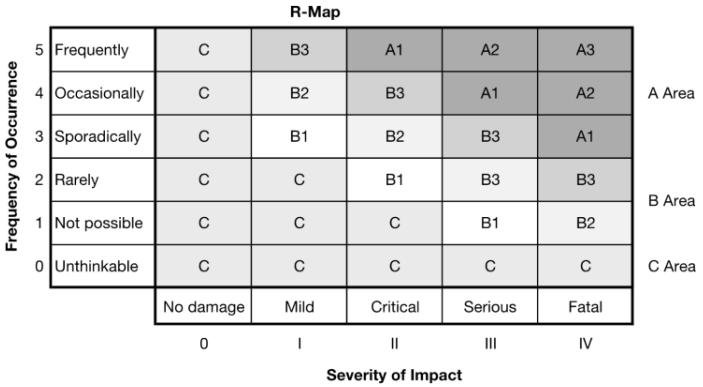

Figure 2. The R-Map

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

Table 2. Axes of The R-Map

(a) Frequency of Occurrence

Level	Qualitative Description	Quantitative Value (cases/unit · year)
5	Frequently	Over 10^{-4}
4	Occasionally	10^{-4} or less and over 10^{-5}
3	Sporadically	10^{-5} or less and over 10^{-6}
2	Rarely	10^{-6} or less and over 10^{-7}
1	Not possible	10^{-7} or less and over 10^{-8}
0	Unthinkable	10^{-8} or less

(b) Severity of Impact

Level	Qualitative Description	Impact to human	Fire damage
IV	Fatal	Death	Fire, building damage
III	Serious	Severe wound, inpatient treatment	Fire
II	Critical	Outpatient medical treatment	Fire on goods, article damage
I	Mild	Slight wound	Smoke on goods
0	No damage	Nothing	Nothing

The R-Map matrix is shown in **Figure 2**. In the vertical axis, there is the frequency of occurrence, and in the horizontal axis there is the severity of impact. The frequency of occurrence uses 0 to 5 of six levels, where the severity of impact uses 0 to IV of 5 levels. The detail is shown in **Table 2**. The risk levels are using the combination of these values in a two dimensional matrix. The matrix is divided into three areas those are A, B, and C. The risk level is raising to the right top of the matrix. The A area is the non-tolerable risk area. If the risk happens during the product development stage, countermeasures should be taken immediately to reduce the risk level, or the product should not be developed any further. The B area is the "As Low as Reasonably Practicable" (ALARP) risk area. Risks in this area should be reduced to the lowest but reasonable and practicable risk as possible, while taking into account the feasibility of risk reduction countermeasure considering risk and benefit, or cost. At last, the C area is the safety area of which the risk level is socially acceptable. Compared with other tolerable risks, the severity of impact and the frequency of occurrence can be considered as low and the risk is considered negligible.

Considering the effectiveness of the R-Map method to assess and to visualize the risk in using and developing a consumer product, in this paper, we propose an adapted R-Map to the software product development especially for the assessment of the risk of software changing. The software changes R-Map is shown in **Figure 3**. As in the original R-Map, the software changes R-Map also uses a two-dimensional matrix. The matrix also consists of the frequency of occurrence in the vertical axis and the severity of impact in the horizontal axis, but instead of using the 6 steps of the frequency of occurrence and the 5 steps in the severity of impact, the software changes R-Map is using simplified 3 steps for each of the axes. The software changes R-Map also defines the risk area in three areas: A, B, and C, depend on the level of the risk. The A areas are having a higher risk of changing compared to B and C areas.

For example, when a given software change has been occurring occasionally and the severity of impact on the existing program is low, then other software functions having the same functionality or characteristic will be having a low risk of changing in the future. But when another given software change is occurring frequently and the severity of impact on the existing program is middle (or high), then other software functions having the same functionality or characteristic will be having a high risk of changing in the future, therefore, appropriate countermeasure and/or pre-caution should be prepared.

For the frequency of occurrence of the software changes we define the qualitative descriptions of each level as "Frequently", "Occasionally", and "Sporadically", representing how often a given software change is occurring during the development life cycle. Here, in order to appropriately define the level of the frequency of occurrence of a given software change, we need to associate a quantitative value to each of those qualitative descriptions. These quantitative values can be obtained from the empirical data which has been collected from the past development projects. In this paper, as already described in the previous section, we determine how often it is for each architectural design element involved in software changes within the software product development life cycle and assign the level of the frequency

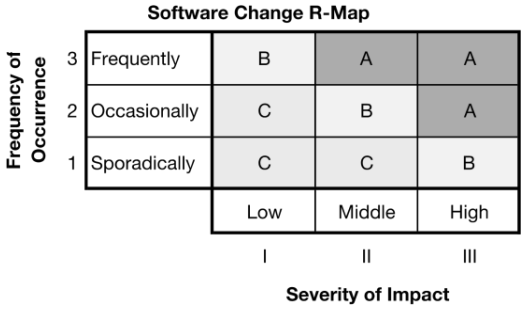

Figure 3. Software Changes R-Map

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

of occurrence of the software changes. The next subsection describes a survey we have conducted in order to obtain this information.

As for the severity of impact of the software changes, we define the qualitative descriptions of each level as "High", "Middle", and "Low", representing how big a given software change has an impact on the existing product. Here, again, as already described in the earlier section, we define how big the impact is by decomposing the impact on the architectural design elements those are "Data Source", "Domain", and "Presentation". This severity of change impact can also be obtained from the empirical data which has been collected from the past development projects. In this paper, we determine the impact of software change based on how many elements have been involved due to the software change by conducting a survey described in the next subsection.

3.3 Survey of The Risk of Changing: A Survey to Obtain the Degree of Volatility of Architectural Design Element

Our proposed method as described in section 3.1 utilizes the deployment tables from customer requirements to the architectural design elements to transfer the degree of volatility of the architectural design elements to the degree of volatility of customer requirements. However, we cannot apply the method without fundamental data to determine the degree of volatility of the architectural design elements. Therefore, we introduced the R-Map as described in section 3.2 in order to obtain the risk of changing of the architectural design elements which is then considered as the degree of volatility. In this section, we describe a survey we conducted as an assessment of risk of changing to obtain the degree of volatility of the architectural design elements.

To conduct the assessment of risk of changing, we use the changes tracking record occurred in the development of a software product. It is an information management system used to manage the documents of specifications. The specifications are divided into three types: ASB, 70000 (TOKU), and Common Part. The specifications and their information are registered in a legacy system but without the documents. The developed system retrieves the specification from the legacy system and associates the appropriate documents, and then registers the data into a separate database. The developed system is a Windows desktop application. Beside using the screen for presentation, the system also provides printed output, sends email, and exports to file. There is a separate small web based application which is used as a complement to the system.

Table 3. Software Changes and Treatments Tracking Record (excerpts)

No	Target Program or Functionality	Change Request	Treat	Changes Element							
				Data Structure	Model	Controller	View				WEB
							Screen	Printed	Email	Files	
1	ASB Registration	Add the feature to open the attached file directly from the list.	Add			✓	✓				
2	ASB Registration	When the [Sales No.] is specified, data is not shown even the data exists in F574201. If the [Delivery Time] is specified, the data is shown.	Confirm								
3	70000 Registration	Add the feature to open the attached file directly from the list.	Add			✓	✓				
4	70000 Registration	When the [Sales No.] is specified, data is not shown even the data exists in F574201. If the [Delivery Time] is specified, the data is shown.	Confirm								
5	70000 Registration	Add the search condition for [Registered Flag].	Add		✓	✓	✓				
6	Common Part Registration	Allow the [Applying Term] entry to be empty.	Mod			✓					
7	Search by Sales No. & Order No.	When click on the Excel attachement, the Pdf attachement is opened rather that the Excel file.	Fix			✓					
8	Search by Sales No. & Order No.	When search by [Order No.], show all lines of "70000 Spec" data.	Mod		✓	✓					
9	Search by Assembly Work Order	When search by [Order No.], show all lines of "70000" documents.	Mod		✓	✓					
78	Common Part Registration	Please add the name of the attended approver(s) in the body of the [Approval Request] email. (To know all the persons to whom the email has been sent)	Add		✓	✓			✓		
79	Common Part Registration	Please add the [Common Sales No.] to the search condition and to the search result list.	Add		✓	✓	✓				
80	Search by Assembly Work Order	When clicking on the [Common Part] icon, the [ASB] Document is shown instead of the [Common Part] image.	Fix			✓					
81	Search by Sales No. & Order No.	When the ambigious search is specified, please allow to enter part of the [Version] search condition too.	Mod		✓						
82	ASB Registration	When the [Registered Document] search condition, when [Include only unregistered document] is specified, please include all documents of which "PDF" attachement is not specified, even other attachements are registered.	Mod		✓						
	Number of changes			4	42	39	44				3
	Percentage of number of changes from the total number of changes			3.0	31.8	29.5	33.3				2.3
	Number of independent changes			0	17 (40.5%)	8 (20.5%)	16 (36.4%)				
	Number of changes involving other elements			4	25 (59.5%)	31 (79.5%)	28 (63.6%)				
	Number of changes involving other elements on the right			4	21 (50%)	7 (17.9%)	N/A				

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

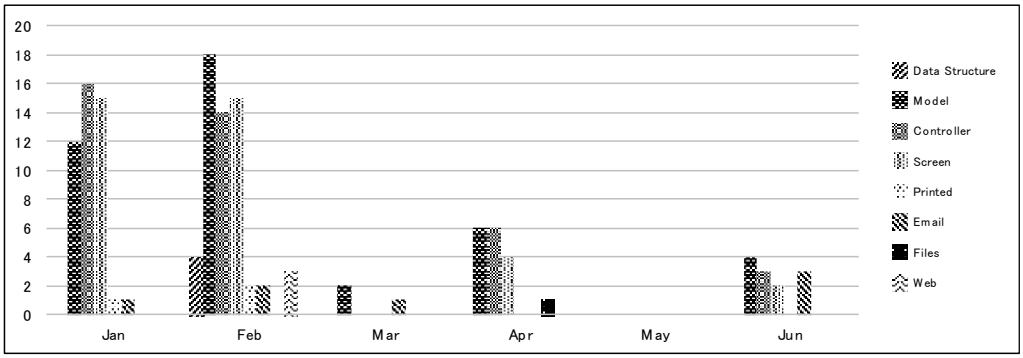

Figure 4. Changes Occurrence Distribution by Time per Architectural Design Element

During the development of the system, and after the release of the system, there are some changes requested by the customer. These changes have been recorded as well as their treatments. In this paper, we will use this record to obtain the risk of changes of the software functions, where in this case, the software architectural design elements. Part of the record of the changes and their treatments are shown in **Table 3**.

As shown in **Table 3**, the changes have been decomposed into their appropriate architectural design elements. The system has been developed considering 3-layer architecture. Data from the database retrieved and updated using a "model" class. This class provides a dataset which is then processed in a class called "controller". This class processes the data and displays the result to the "screen". Other than to the screen, the result may be displayed as a printed matter, email, or files. Here we decompose the architectural design elements into four groups: data structure, model, controller, and view which consists of screen, printed, email, and files. The last element, the WEB, is a web application which is a complement to the system and we consider it as a separate subsystem.

First, we will obtain the severity of impact when a software change has been conducted. We obtain this by finding out how each change request has an impact on each architectural design element. We count the number of changes on one element which has an impact on other elements. If the implementation of the change request involves program changes in all elements, the change is considered having many regressions in changes. In the other case, if the implementation only involves program changes in one independent element, the change is considered having few regressions in changes. If the implementation involves program changes in two or more elements, the change is considered having an average regression in changes. We can see in **Table 3** that changes occurred on the data structure element always involve all the other elements, so we can consider that change on this element is having high impact on the software. Also, as we see that changes occurred on the model element, half of them, involve the rest of other elements, we can consider that change on this element is having high or above middle impact on the software. The next element of which the changes are giving relatively middle impact on the software is the controller element. And finally, the changes on view element are having the lowest impact on other elements. By substituting the model (and data structure), controller, and view elements to data source, domain, and presentation elements, we put the result of the severity of changes impact of these elements in **Table 4**.

Next, we obtain the frequency of changes occurrence of each architectural design elements and determine the level. However, if we see the numbers of occurrence in **Table 3**, except for the data structure element, all other elements are having relatively the same number, which makes us difficult to assign the level. To gather more information about the characteristic of the changes record, we decided to see the distribution of changes by time per element. The graph is shown in **Figure 4**.

As shown in the graph, we can see that change requests are occurred frequently in the beginning of the tracking timespan. Change requests begin, mostly, with changes on the controller element, followed with screen, and model elements. In the next period of month, there is a huge changes occurred on the model element, where if we study the detail of the changes, most of them have been involved by the changes on the data structure. Accordingly, changes on the controller and view elements were also the impact of changes on the data structure. Furthermore, when we study the detail of change requests of which the changes occurred on the model element, most of them are corrections of the specification given at the beginning of development. So, if considering the frequency of occurrence in the data source architectural design element, it should mainly involve changes on the data structure. Therefore, considering the result of the changes record, as well as the distribution of changes occurrence by the time of occurrence, we can determine

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

Table 4. Risk of Changing of the Architectural Design Element

Architectural Design Element	Severity of Change Impact	Frequency of Change Occurrence	Risk of changing	Risk of changing (ordinal scale)
Data Source	III	1	B	2
Domain	II	3	A	3
Presentation	I	2	C	1

the level of frequency as frequently for the domain element, occasionally for the view element, and sporadically for the data source element. We put the result of the frequency of occurrence of these elements in **Table 4**.

Finally, after we finished defining the level of severity of change impact and the level of frequency of change occurrence, we can then obtain the risk of changing of each architectural design element using the software changes R-Map shown in **Figure 2** and put the result in **Table 4**. The ordinal scale value of risk of changing of each architectural design element is obtained from the risk level in the R-Map, where A area has a bigger scaled value than C area. In this paper, we use ordinal scale values of 1, 2, and 3 which is associated with C, B, and A, respectively, and use it as the degree of volatility of the architectural design element.

4 Case Study: Software Requirements Volatility Analysis in the Development of an Active and Running Information System

In this section, we describe a case study we conducted in order to show how we apply our proposed method and to analyze the result. The software product we use for our case study is an information system used in new student admission at Institute of Statistics (or Sekolah Tinggi Ilmu Statistik in local language), a college located in Jakarta Indonesia. The recruitment of new student is conducted once every year, targeting at high school graduates from all over the country. Until the year of 2009, the application for the admission is conducted manually. The applicant had to come to the nearest office of Statistics Indonesia located in the capital city of each province where the institute receives and processes the application. Indonesia is a big country and the applicants are growing every year. To reduce the overall cost of admission process and to give more opportunity to them who want to apply for the admission, an on-line admission system has been proposed, built and then started for its operation in the year of 2010.

We conducted applying our proposed method of software requirements analysis using QFD considering the requirements volatility from the risk management point of view. We first conduct the deployment of customer requirements and derive the relationship between software functions and architectural design elements. Then, we use the risk of changing of the architectural design elements obtained from the software changes R-Map, and consider it as the degree of volatility of the architectural design elements. We then, while referring to the relationship between architectural design element and the software functions, we calculate the degree of volatility of the software functions, and furthermore, using the results, we calculate the degree of volatility of the customer requirements, and finally, we analyze the overall results.

Table 5. Deployment of Customer Requirements to Software Functions (excerpt)

Customer requirements		Software functions	
CR ID	**Level 2**	**SF ID**	**Software functions**
CR1.1	Applicant enter their form on-line.	SF1.1.1	Applicant verifies his/her email address.
		SF1.1.2	System sends email confirmation that contain the link for registration.
		SF1.1.3	Applicant use the link included in the confirmation email and the system displays the content of the registration form.
		SF1.1.4	Applicant enter/edit the form on-line.
		SF1.1.5	Applicant submits his/her data.
CR1.2	Applicant will get confirmation after registration on-line.	SF1.2.1	After registration, there is a confirmation displayed to the applicant.
		SF1.2.2	After registration, system send a confirmation via email.
CR2.1	Prerequisites for application are validated on-line before submiting the application.	SF2.1.1	System will validate the form entry using prerequisite rules during the entry.
		SF2.1.2	System will validate, again in the server, when applicant send the form entry to the server.
		SF2.1.3	If the validation failed, applicant will get notice and will be able to continue the entry.
CR3.1	Applicant pays for the submission fee via Bank.	SF3.1.1	After submitting the entry form, system will generate a token for bank payment and send it along with confirmation email.
		SF3.1.2	Applicant pays the submission fee via bank (*bank system*'s *functionality*).
CR3.2	Application data will be reflected as payed after the payment.	SF3.2.1	System will communicate with the bank to receive the payment information.
CR4.1	Applicant will get his/her admission ticket, on-line, after his/her payment for the submission fee is confirmed.	SF4.1.1	Applicant logins into his/her registration
		SF4.1.2	Application down
CR5.1	Applicant will get the examination's result on-line on the scheduled date.	SF4.1.1	^

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

Table 6. Calculation of Customer Requirements Degree of Volatility

Customer requirements		Software functions																								Degree of Volatility	Degree of Volatility ratio (%)
CR ID	Level 2	SF1.1.1	SF1.1.2	SF1.1.3	SF1.1.4	SF1.1.5	SF1.2.1	SF1.2.2	SF2.1.1	SF2.1.2	SF2.1.3	SF3.1.1	SF3.1.2	SF3.2.1	SF4.1.1	SF4.1.2	SF5.1.1	SF6.1.1	SF6.1.2	SF7.1.1	SF7.1.2	SF7.1.3	SF8.1.2	SF9.1.1	SF10.1.1	SF11.1.1		
CR1.1	Applicant enter their form on-line.	◎	◎	◎	◎	◎																					255	15.6
CR1.2	Applicant will get confirmation after registration on-line.						◎	◎																			110	6.7
CR2.1	Prerequisites for application are validated on-line before submiting the application.								◎	◎	◎																200	12.3
CR3.1	Applicant pays for the submission fee via Bank.											◎	△														95	5.8
CR3.2	Application data will be reflected as payed after the payment.													△													23	1.4
CR4.1	Applicant will get his/her admission ticket, on-line, after his/her payment for the submission fee is confirmed.														○	◎											138	8.5
CR5.1	Applicant will get the examination's result on-line on the scheduled date.														○		◎										98	6.0
CR6.1	Organizer can monitor on-line in a realtime the progress of submission.																	○	◎								86	5.3
CR7.1	Organizer can modify the individual data when requested by the applicant.																	○		○	◎	◎					185	11.3
CR8.1	Applicant can modify his/her own data before the payment for the submission fee is made.				◎	◎									○								◎				178	10.9
CR9.1	Organizer enter the result of examination into the applicant's data by batch processing.																							△			10	0.6
CR10.1	Organizer can get or print the list of all applicants.																	○							◎		126	7.7
CR11.1	Organizer can get or print the list of all successful candidates.																	○								◎	126	7.7
Degree of Volatility		7	17	11	5	11	5	17	18	16	6	17	10	23	11	21	13	7	13	13	14	11	13	10	21	21		
Degree of Volatility ratio (%)		2.1	5.1	3.3	1.5	3.3	1.5	5.1	5.4	4.8	1.8	5.1	3.0	6.9	3.3	6.3	3.9	2.1	3.9	3.9	4.2	3.3	3.9	3.0	6.3	6.3		

Before we start the first step of the proposed method, we prepared the customer requirements by conducting the deployment of customer requirements in two levels of abstraction. We use the list in level 2 of the customer requirements as the input of the proposed method.

The first step, STEP 1, is to deploy the customer requirements and derive the software functions as well as the relationship between both items. The deployment of software functions is shown in **Table 5**, and the relationship between customer requirements and software functions is shown in **Table 6**. Please ignore the columns and rows data containing a degree of volatility for now. We will explain them later.

From **Table 5**, we can see that some customer requirements have been deployed to use the same software function to fulfill them. For example, for the CR1.1 "Applicant enter their form on-line" and CR8.1 "Applicant can modify his/her own data before the payment of the submission fee is made.", as both requirements are involving the function to edit the form on-line, they are using the same software function, the SF1.1.4 "Applicant enters/edits the form on-line". In this case, we only deploy the function "Applicant enters/edits the form on-line" once.

Table 6 shows the relationship between customer requirements and software functions. This table shows how a customer requirement is fulfilled with one or more software functions. The relationship shows how strong a customer requirement is related to the software functions, whether the function has the main functionality, or it just provides a supporting functionality, where the first will have a stronger relationship than the other. As mentioned in section 3.1, we use three levels of relationships described as strong, moderate, and weak, using symbols of "◎", "○", and "△", with assigned values of 5, 3, and 1, respectively. For example, the software function SF4.1.1 which provides functionality for applicant to login to his/her registration page provides only for supporting functionality because the applicant will conduct other function which has the main functionality.

In the next steps, STEP 2 and STEP 3, we use the list of software functions and determine their relationship with architectural design elements. In this case study, we use the 3-layer architecture which consists of data source, domain and presentation layers as the elements of the architectural design. In addition, from the derived software functions, we identify that there is an external system, the bank payment system, related to the main system. We have decided to treat this external system as an independent element of architectural design. **Table 7** shows the deployment table and the relationship between software functions and architectural design elements.

As in the previous step, the relationship between software functions and architectural design elements is determined by considering how strong is the cohesion of the software function to the architectural design element. This consideration is better conducted at the level of program design. For example, for the software function SF1.1.1 "Applicant verifies his/her email address"' is related to the presentation layer and the data source layer. Considering that the

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

Table 7. Calculation of Software Function Degree of Volatility

Software functions		Architectural Design Elements				Degree of Volatility	Degree of Volatility ratio (%)
SF ID	Software functions	Presentation	Domain	Data source	External		
SF1.1.1	Applicant verifies his/her email address.	◎		△		7	2.1
SF1.1.2	System sends email confirmation that contain the link for registration.		◎	△		17	5.1
SF1.1.3	Applicant use the link included in the confirmation email and the system displays the content of the registration form.	◎		○		11	3.3
SF1.1.4	Applicant enter/edit the form on-line.	◎				5	1.5
SF1.1.5	Applicant submits his/her data.	△		◎		11	3.3
SF1.2.1	After registration, there is a confirmation displayed to the applicant.	○		△		5	1.5
SF1.2.2	After registration, system send a confirmation via email.		◎	△		17	5.1
SF2.1.1	System will validate the form entry using prerequisite rules during the entry.	○	◎			18	5.4
SF2.1.2	System will validate, again in the server, when applicant send the form entry to the server.	△	◎			16	4.8
SF2.1.3	If the validation failed, applicant will get notice and will be able to continue the entry.	○	△			6	1.8
SF3.1.1	After submitting the entry form, system will generate a token for bank payment and send it along with confirmation email.		◎	△		17	5.1
SF3.1.2	Applicant pays the submission fee via bank (*bank system's functionality*).				◎	10	3.0
SF3.2.1	System will communicate with the bank to receive the payment information.		◎	△	○	23	6.9
SF4.1.1	Applicant logins into his/her registration page.	△		◎		11	3.3
SF4.1.2	Application downloads his/her admission ticket.		◎	○		21	6.3
SF5.1.1	Applicant will see the examination's result.	○		◎		13	3.9
SF6.1.1	Organizer logins into the special site for organizer.	△		○		7	2.1
SF6.1.2	Organizer views the list of submissions.	○		◎		13	3.9
SF7.1.1	Organizer selects the applicant by entering his/her submission number.	○		◎		13	3.9
SF7.1.2	Organizer edits the applicant's data.	◎	○			14	4.2
SF7.1.3	Organizer submits the applicant's data.	△		◎		11	3.3
SF8.1.2	Applicant views his/her data.	○		◎		13	3.9
SF9.1.1	Organizer, with its IT staff, imports the result of examination into the database.			◎		10	3.0
SF10.1.1	Organizer download the list of all applicants.		◎	○		21	6.3
SF11.1.1	Organizer download the list of all successful candidates.		◎	○		21	6.3
	Degree of Volatility	1	3	2	2		

implementation of this function in the level of program design is mainly in the presentation layer, then we determine the function has a strong relationship with the presentation layer. As for software function SF2.1.2 "System will validate, again in the server, when applicant sends the form entry to the server."', we determine the function has a strong relationship with the domain layer, because the function is mainly doing logical processing for validation.

And finally the last two steps, STEP 4 and STEP 5, are to use the result of the degree of volatility of architectural design elements to obtain the degree of volatility of software functions and furthermore the degree of volatility of customer requirements. **Table 7** shows the result of the calculation of the degree of volatility of software functions, and **Table 6** shows the result of the calculation of the degree of volatility of customer requirements.

In **Table 7**, we use the degree of volatility of architectural design elements obtained from the software changes risk assessment. We use the risk of changing (ordinal scale) from **Table 4**, and consider it as the degree of volatility of the architectural design elements, that is 1 for Presentation, 2 for Data Source, and 3 for Domain; and for the External element, we assign the value of 2 assuming the risk of changing of this element is on average. The degree of volatility of software functions obtained in the table then is used in **Table 6** to calculate the degree of volatility of customer requirements.

From **Table 6** we can see which customer requirements are having a low ratio of the degree of volatility and which ones are having a high ratio of the degree of volatility. For example, for the customer requirement CR3.1 "Applicant pays for the submission fee via Bank." which has a low ratio of the degree of volatility, it only mentions that the payment should be possible to be made via bank, and further specification of the payment does not involve the customer rather than the bank. As for customer requirements CR10.1 and CR11.1 which provide the functionality to download a list of applicants or successful candidates and also have a low ratio of the degree of volatility, because they only involve the data source and the domain layers, even if there is a change in the list, the change does not give a wide impact on the development. And lastly, for the customer requirements CR1.1, CR2.1, CR7.1 and CR8.1, which all involve the functionality to enter, edit and validate the application on-line and all have a high ratio of the degree of

Proceedings of The 22nd International Symposium on QFD, Boise, Idaho USA, 2016
ISBN 1-889477-27-3

volatility, they are strongly related to the presentation layer. Item in the application form may be changed, validation rule may be changed, and prerequisites may also be changed. Therefore, these customer requirements need to be investigated deeply, or, in addition, these requirements should be designed and implemented using loose architectural design in order to anticipate the future changes.

5 Concluding Remarks

Previous studies reported that software requirements volatility cannot be identified until the phase of software design [5]. In this research, we proposed a systematic method to conduct analysis of software requirements volatility. By employing QFD method, we can deploy software requirements to software functions and architectural design elements subsequently. Then, after conducting software changes risk assessment using R-Map we can obtain the risk of changing which is then considered as the degree of volatility of the architectural design elements. The degree of volatility of software functions and software requirements can then be calculated subsequently. From the case study we conducted, the results show that we can expect that the values of the degree of volatility of the software requirements obtained using the proposed method represent the software requirements volatility.

In order to refine the result of risk assessment as well as the requirements volatility analysis, more studies and experiments should be conducted.

Acknowledgement

We would like to thank to Naofumi Takayama from Sync Information Systems, Inc. located in Kofu, Japan, and to Institute of Statistics (Sekolah Tinggi Ilmu Statistik) located in Jakarta, Indonesia, both for providing the data of the software requirements specification including the changes record during the development life cycle.

References

[1] H. Dev and R. Awasthi, "A Systematic Study of Requirement Volatility during Software Development Process", *International Journal of Computer Science Issues*, vol. 9, no. 2, pp. 527–533, 2012.

[2] Japanese Standards Association, "JIS Q 9025:2003 Performance Improvement of Management Systems – Guidelines for Quality Function Deployment," 2003.

[3] N. Inoue, "Application of R-Map in Product Development" (in Japanese) in *NTT Facilities Research Institute Annual Report*, No. 26, pp. 57–63, 2015.

[4] B. Sharif, S. A. Khan, and M. W. Bhatti, "Measuring the Impact of Changing Requirements on Software Project Cost: An Empirical Investigation", *IJCSI International Journal of Computer Science*, vol. 9, no. 1, pp. 170–174, 2012.

[5] N. Nurmuliani, D. Zowghi, and S. Fowell, "Analysis of Requirements Volatility during Software Development Life Cycle", *Proceedings of the 2004 Australian Software Engineering Conference ASWEC04*, pp. 28–37, IEEE, 2004.

[6] S. L. Lim and A. Finkelstein, *Anticipating Change in Requirements Engineering*, pp. 17–34. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[7] ISO/IEC and IEEE, "ISO/IEC/IEEE 24765:2010 - Systems and software engineering – Vocabulary", vol. 2010, p. 410, 2010.

[8] F. Bushmann, R. Meunier, H. Rohnert, P. Somerlad, and M. Stal, *Pattern-oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 2001.

[9] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, *Patterns of Enterprise Application Architecture*, Addison Wesley, 2002.

[10] T. Kishi, N. Noda, and Y. Fukasawa, *Software Architecture (in Japanese)*. Kyoritsu Shuppan, 2005.

[11] K. Brown, G. Craig, G. Hester, D. Pitt, R. Stinehour, M. Weitzel, J. Amsden, P.M. Jakab, and D. Berg, *Enterprise Java Programming with IBM WebSphere*, Addison Wesley, 2001.

[12] Y. Anang and Y. Watanabe, "Applying layering concept to the software requirements analysis and architectural design", *Joint Proceedings of REFSQ-2016 Workshops co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016)*, pp. 45–50, 2016.

[13] S. Haag, M. K. Raja, and L. L. Schkade, "Quality function deployment usage in software development", Commun. ACM, vol. 39, pp. 41–49, 1996.
[14] G. Herzwurm, S. Schockert, and W. Pietsch, "QFD for Customer-Focused Requirements Engineering", *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pp. 330–338, 2003.
[15] M. Gloger, S. Jockusch and N. Weber, "Using QFD for Assessing and Optimizing Software Architectures the System Architecture Analysis Method", *Proceedings of the 5th International Symposium on Quality Function Deployment*, pp. 119–127, 1999.
[16] K. Matsumoto (translated by Chong Zhao), "Risk Management and the R-Map", *Official Magazine of the New Zealand Organization for Quality*, pp. 6–10, 2012.
[17] R-Map Practice Workshop, Union of Japanese Scientists and Engineers (JUSE), "Introduction to R-Map for Product Safety and Risk Assessment" (in Japanese), https://www.juse.or.jp/reliability/introduction/03.html, 2011.
[18] Y. Sakai, "Validation Method for Embedded Software Product Line" (in Japanese), *Proceedings of Japan Symposium on Software Testing 2004*, 2004.
[19] Y. Kyoya, T. Nakano, K. Noguchi, T. Nishioka, "QFD for Software Development Considering Future Risk", *Proceedings of 15th Symposium on QFD / 9th International Symposium on QFD*, pp. 257–274, 2003.
[20] ISO, "ISO 16355-1:2015 - Application of statistical and related methods to new technology and product development process -- Part 1: General principles and perspectives of Quality Function Deployment (QFD)", 2015.
[21] The Ministry of Economy, Trade, and Industry, Japan, "Risk Assessment Handbook for Customer Product—First Edition" (in Japanese), May, 2010.
[22] The Ministry of Economy, Trade, and Industry, Japan, "Risk Assessment Handbook—Practice" (in Japanese), 2011.

Yunarso Anang received B.E. degree in 1995 and M.E. degree in 1997 both in software engineering from University of Yamanashi, Japan. He was with SYNC Information System, Inc., Japan from 2000 to 2007, as a senior system engineer. From 2008, he is with Institute of Statistics (Sekolah Tinggi Ilmu Statistik), Jakarta, Indonesia, as a lecturer in computer science. From 2014, he is a Ph.D. student at University of Yamanashi, Japan. His research interests include software engineering and quality function deployment.

Masakazu Takahashi received B.S. degree in 1988 from Rikkyo University, Japan, and M.S. degree in 1998, Ph.D. degree in 2001, both in Systems Management from University of Tsukuba, Japan. He was with Ishikawajima-Harima Heavy Industries Co., Ltd. from 1988 to 2004. He was with Shimane University from 2005 to 2008. He is a professor in University of Yamanashi, Japan since 2014. His research interests include software engineering and safety.

Yoshimichi Watanabe received the B.S. and M.S. degrees in computer science from University of Yamanashi, Japan in 1986 and 1988 respectively and received D.S. degree in computer science from Tokyo Institute of Technology, Japan in 1995. He is presently an associate professor of the Department of Computer Science and Engineering at University of Yamanashi. His research interests include software development environment and software quality.