Collaborating Acceptance Test-Driven Development and QFD in Business System Software Development

Yunarso Anang
University of Yamanashi
4-3-11 Takeda, Kofu, Yamanashi
400-8511 Japan
+81(55)2208651
g14dma01@yamanashi.ac.jp

Masakazu Takahashi
University of Yamanashi
4-3-11 Takeda, Kofu, Yamanashi
400-8511 Japan
+81(55)2208651
mtakahashi@yamanashi.ac.jp

Yoshimichi Watanabe
University of Yamanashi
4-3-11 Takeda, Kofu, Yamanashi
400-8511 Japan
+81(55)2208651
nabe@yamanashi.ac.jp

ABSTRACT
One of the challenges in developing a high-quality business system software is the ability to accommodate constantly changing user requirements. Unlike the development of a tangible product like mechanical or electrical parts, software's very own nature allows it to be changed to provide different or new functionality. However, constantly changed software can degrade its quality by introducing new problems and loose its maintainability. Test Driven Development (TDD) is a software development practice, which enables developers to design the software so the codes are loosely coupled and testable thus have high quality in the term of maintainability. Acceptance Test Driven Development (ATDD) is an extension of TDD. While TDD answers the programmer's needs, ATDD is a technique for enhancing communication with non-programmer people which most users are. In this paper, we describe how we introduce QFD into ATDD to derive the acceptance tests from user requirements, and how to break down into the lower level of test cases. Our goal is to provide a technique to develop software with user requirements in mind and high maintainability. To evaluate our approach, we simulate our approach to the development of a real business system which has already been developed using TDD approach.

KEYWORDS
Acceptance Test-Driven Development, QFD, Business system, Software development

1. INTRODUCTION
Enterprise business system, or in short business system, is defined as a system used to process the primary function which is essential for a company to perform its business[1]. The primary function includes, for example, accounting for a banking company or production, sales, or inventory management in a general company. Most of the business systems cover functions in the entire company. For a common set of functions, an ERP (Enterprise Resource Planning) package software may become the right choice, but for a domain specific feature, most companies still prefer to build the system from the scratch.

One of the challenges in developing a high-quality business system is the ability to accommodate the constantly changing user requirements which still commonly exists in the development of large and complex software projects [1]. The factors contributed to the change in requirements are such as the diversity of needs among different customers, misconception of the application domain by the development team, or in general, poor communication between users, customers, stakeholders, and developers [2]. The user refers to the end-user who uses the software product and the customer refers to a group of users, an organization or a unit of an organization who owns and/or uses the software product. The tendency of requirements to change over time in response to the evolving needs of customers, stakeholders, organization and work environment was then used to define the requirement volatility [3]. Unlike the development of a tangible product like mechanical or electrical parts, software's very own nature allows it to be changed to provide different or new functionality. However, constantly changed software can degrade its quality by introducing new problems and loose its maintainability.

Test Driven Development (TDD) is a software development practice, where instead of creating the test after the development, the test (preferably automated) is created before starting the development of the production code [4]. The

[1] IT keyword dictionary (in Japanese) http://www.sophia-it.com

production code refers to the source code which does the functionality of the software product, while the test code is the source code which is intended to test the functionality of the production code. TDD, rather than a regular testing technique, is a design approach and thus it has not only the influence on the external quality of the software product but also on the internal quality of the source codes [5]. Like TDD, Acceptance Test-Driven Development (ATDD) is also a practice in the software development which involves creating tests before writing production codes [6]. Acceptance tests, also called customer tests, are tests specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer [7]. Acceptance tests are derived from user stories that describe required output, features, and functionality for software to be built. While the test codes in TDD represent the expectations of the functionality of the software, the tests in ATDD represent expectations of the behavior of the software. ATDD describes the tests based on the behavior of the user which is called user stories or feature which is first introduced in the concept of Behavior-Driven Development (BDD) [8].

This paper discusses our preliminary result of the study on the understanding of how the result of Quality Function Deployment (QFD) can be used or combined with the input of the Acceptance Test-Driven Development (ATDD) in the development of a complex software system such as a business system. In this paper, we describe how to collaborate the QFD on the development of the software product which is driven by acceptance tests.

The rest of this paper is organized as follows. Section 2 describes related studies. Section 3 describes the proposed development method. The section also gives an example of the implementation of the acceptance tests in the development of a real business system. Finally, we conclude this paper in Section 4.

2. RELATED STUDIES

This section describes the literature and the studies related to ATDD, TDD, and the application of QFD in the software development which is related to ATDD.

2.1 Acceptance Test-Driven Development

ATDD is a development practice used typically in conjunction with BDD. The developer team starts collecting the needs of the customer by gathering all possible scenarios on the target business. As described in [6], those scenarios represent examples of tasks or behaviors derived from the business goals. The developer team defines the scenario using the form of the user story. As explained in [8], a user story consists of the definition of how (1) a person or role (2) performs a feature (3) so that the person benefit of something; and an acceptance test consists of the definition of when (4) some initial context is given (5) under some conditions, then (6) how to ensure the outcomes. Where there is no real benefit for a story, the part (3) can be omitted. For one story, there are several acceptance tests or scenarios can be considered. Figure 1 shows an example of user story representing one task of listing companies in a text control by entering some characters of the beginning of the company name. The user story is then formulated as a set of executable acceptance tests before the production code is written. Therefore, the production code is validated against the acceptance tests, rather than against the developer's interpretation of the requirement.

	Feature:	Search for a company
(1)-(2)		As an operator, I want to select a company.
	Scenario:	Search for a company with name starting with “Fuj”
(4)		***Given*** A text control
(5)		***When*** I enter “Fuj”
		And I press the [Enter] key
(6)		***Then*** I get the list of companies having the name starting with “Fuj”.

Figure 1: An example of user story and acceptance test

There are few studies related to ATDD. Kroizer suggests the use of ATDD integrating validation testing in Scrum giving the result of analysis with the actual data [9]. Because the acceptance tests implement the criteria which are based on the customer's requirements, they give quick feedback about the status of a story, whether pass or fail. Because the acceptance tests are automatic, they are easy to re-run and re-used to ensure the regression test will pass in the next sprints, where sprint is the basic unit of development in Scrum. However, Kroizer also suggests not to use ATDD for short projects when there is no intention to re-use the automation because the drawback for the time and effort necessary to learn the tool and to maintain the automation is worth nothing. The use of automation tests should also not to be overrated because the test code itself may contain bugs and may not cover all functionalities. Manual tests should be used for depth tests or to cover-up potential bugs.

Most recently, a study on applying ATDD to the construction of cryptographic software has been conducted [10]. The study shows how the ATDD uses the predefined test vectors as part of the user story. Test vectors are data sets constructed with the aim of evaluating the correctness of cryptographic implementations, not their security. Each test vector consists of key(s), plain text, and cipher text. Then for each encryption method, the user story can be constructed using the encryption method as the role and the rest of parameters as the remaining parts of the user story.

While both studies demonstrated quite well the benefit of applying ATDD, there is no argument regarding of how to consider the quality measurement rather than the functionality assurance.

2.2 QFD and Software Development

QFD has been accepted as an effective and promising technique in alleviating the problems associated with the early phases of requirements and specifications [11]. The unique potential of QFD will be a catalyst for rapid technology deployment and for parameter design of information systems and as a method for specification of the "right product" where most software developers are still struggling for nowadays. Watanabe et al. proposed a method of requirements analysis using QFD, where the concept of user scene and goal to catch the customer need has been introduced [12]. Furthermore, Anang et al. proposed a method to apply QFD in the intertwined processes of requirements analysis and architectural design [13]. Herzwurm et al. stated that in the development of software product QFD can bridge the gap between software development and the customer by transferring the customer needs to concrete product requirements [14]. They also stated the greatest advantages of applying QFD to the software development is its inherent flexibility. The QFD can be embedded in incremental and iterative development to a growing extent, which also intimates its integration with agile development approaches like Scrum, a framework for developing sustaining complex products [15], [16].

There are few studies of QFD related to software development which mentions the acceptance test. Nayar et al. developed an algorithm for creating the house of quality matrix in their proposed approach for the integration of XP programming and QFD [17]. However, how the user stories and the acceptance tests are deployed into the development is not explicitly described.

2.3 The True Nature of QFD

Ohfuji states that performing QFD is not about creating the quality table, which is also referred to as house of quality, or other two-dimensional tables, instead applying the following three principles: the principle of decomposition and integration, the principle of multi-dimensional development and visualization, and the principle of the consolidation and breakdown, in the actual scenes [18]. The true nature of applying QFD is about applying these principles, and e.g. the quality table is only one example of applying the principles. Shindo also describes that as the true nature of QFD is to capture the scene in multidimensional aspects, it can also be used as a method for problem-solving, especially for an intangible product like software [19]. The method proposed in this paper applies these principles.

3. THE PROPOSED QFD-ATDD DEVELOPMENT METHOD

This section describes the proposed method of collaborating QFD and ATDD. First, it describes the method to deploy the customer requirements into user stories which are explained using an illustration. Next, it describes how to reflect the user stories into the development of software products. And finally, it describes an example of the implementation of the acceptance tests in the development of a real business system.

3.1 From Customer Requirement to User Stories

Typical QFD tasks start with the task of hearing the needs from the customer. The goal of this task is to gather information about what does the customer want from the product. The typical sentences of the customer need are such as "We want this product can do this." or "We want this product can do this smoothly.". The first form is a functional need and the latter is also a functional need but with a non-functional attribute of quality. Those needs are then decomposed into more detailed needs which act as the customer requirements. These customer requirements are then converted into quality characteristics and furthermore are deployed into functional components and other elements of the process. As a software become the target of the QFD, the tasks are typically the same. The challenge is how to treat the user behavior in correspond to the user requirement.

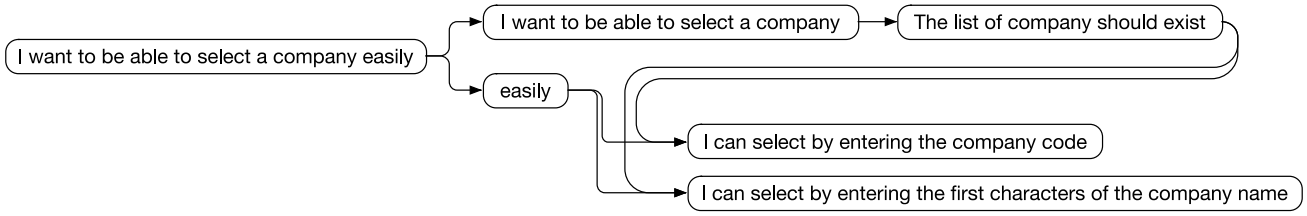

Figure 2: Deriving detailed user requirement

For example, there is a given user requirement that says "I want to be able to select a company easily.”. This requirement can be divided into a functional requirement that says "I want to be able to select a company" and a non-functional requirement that says "easily". Following the first principle of QFD, this requirement needs to be decomposed into detailed functional and non-functional requirement. In this example, in order for the user to be able to select a company easily, the list of companies should exist, somehow, and then the user can select the company from the list. Furthermore, to make even easier if there are a lot of companies in the list, the user can select the company by entering the company code (if there is) or by entering the first characters of the company name to make the list shorter and make the selection easier. This

task of deriving detailed user requirement is illustrated in Figure 2. From this example, we can then derive and compose the following feature and user stories. The feature is to select a company and the user stories are as shown in Figure 3.

Figure 3 shows that for the user requirement in the above example, there are two scenarios of usage with the same input (the *Given*), two different conditions (the *Whens*), and the two types of result (the *Thens*). In this case, the operator is the person (the *role*) conducting this operation. And in this example, there is one non-functionality attribute that is "easily", which in this case, the non-functionality attribute has been converted into two behaviors or operations.

Feature: Select a company
As an operator, I want to select a company.
Scenario: Search for a company with the code is “F01”
Given A list of companies
When I enter the company code “F01”
Then I get the company which code is “F01”.
Scenario: Search for a company with the name is starting with “Fuj”
Given A list of companies
When I enter “Fuj” as part of the company name
Then I get the list of companies having name starting with “Fuj"

Figure 3: Derived user stories for the user requirement

From the given example, the first two principles of QFD have been applied. The user requirement has been divided into multiple elements and dimensions to make the requirement more concrete. After that, those elements are then combined to compose the features along with user stories representing the given requirement. The process and the result have been visualized using block diagram and user story card. The process of converting the user requirement into user stories is summarized using a block diagram in Figure 4.

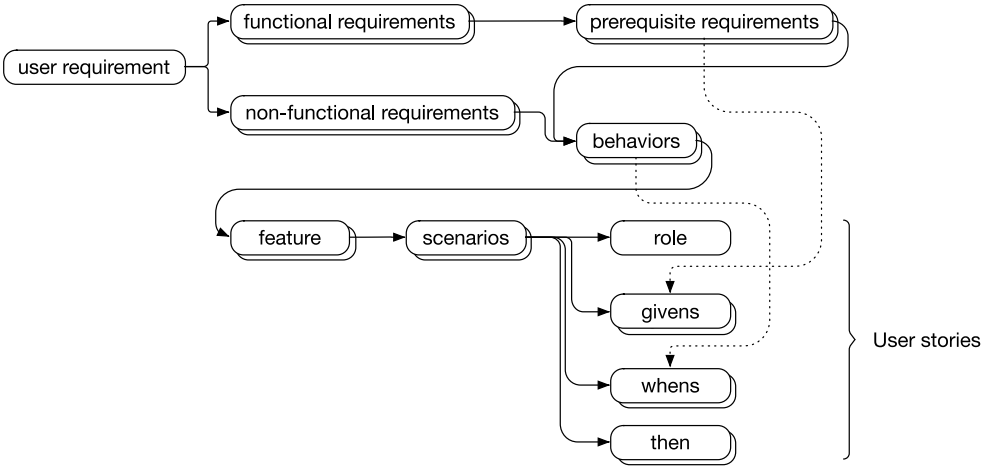

Figure 4: User requirement to user stories

As shown in Figure 4, the prerequisites requirements of performing the behaviors and the behaviors themselves compose the user stories as the *Givens* (the initial context) and *Whens* (the conditions) respectively. Each of this context and conditions may be re-used in different scenarios. The same scenario of the user stories may consist of multiple test cases of conditions to ensure the functionality of the feature. These test cases should also be defined not by developer team based on their interpretation, but by the customer together with the developer team.

After the user stories are derived from the user requirement, as indicated in ATDD, the user stories are then implemented as test codes for acceptance tests. The implementation of the acceptance tests, and furthermore the production codes, is the job of the developer team alone. However, as the production codes are validated against the acceptance code, which means the production codes are validated against the user requirements.

3.2 From the User Stories to the Development

This section describes the tasks of implementing the user stories into the acceptance tests, and furthermore into the production codes. The tasks involve two major tasks: the implementation of the acceptance tests from the user stories, and the implementation of the production codes to make the acceptance tests pass. The latter is typically the practice of TDD. In this paper, an implementation based on object oriented programming is used to describe the tasks.

A feature, which practically consists of user stories, is implemented in one feature class. Each scenario in the user stories is implemented as a method in the feature class. The scenario method then runs the test class and call each of the methods associated with *given*, *when*, and *then* from the user story passing each corresponding value. Figure 5 shows the transitions

from the feature to the feature class and to the acceptance test subsequently. Practically, the feature class and also the acceptance class can be generated automatically from the feature definition using a software tool. Doing this could prevent any mistaken by the developer team when transcribing the feature definition into the test codes.

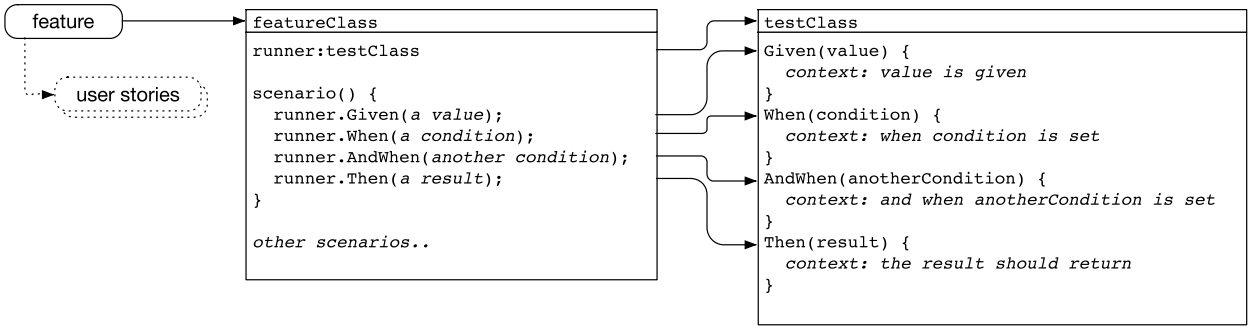

Figure 5: Generic implementation of acceptance tests from a feature

After the test class has been generated, the developer team is ready to write the production codes to make the acceptance tests pass using the TDD technique. This paper does not describe the detail of applying the TDD in the development of complex software such as a business system. We have conducted other parallel study regarding this. Figure 6 gives an illustration of how the production codes are being written in correspond to the acceptance tests.

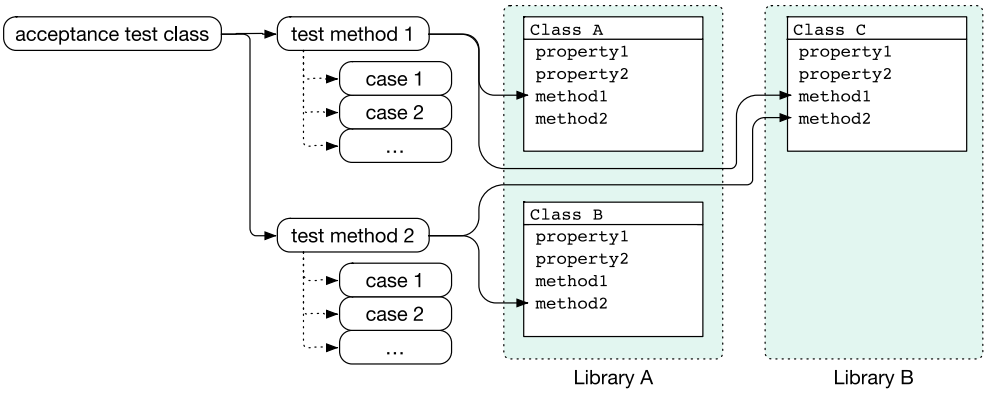

Figure 6: An example of implementation of production codes from an acceptance test

Finally, as shown in the figures in this section and in the previous section, all elements can be managed into the list and the relationship between those elements can be represented using two-dimensional tables as commonly used when practicing QFD. However, instead of using the conventional form of a two-dimensional table such as that used in a spreadsheet, that information can be stored in a relational database as indicated by Ohfuji [20]. By storing the information in a database, every member of the project, including the developer team and the customer, can transparently view how the product is built and trace every element forth and back. This mechanism was left as a future work.

3.3 An Example

This section gives an example of the implementation of the acceptance tests in the development of a real business system. The system is used at a manufacturing company and as part of their enterprise system. The system used in this evaluation itself is used to manage the return and disposal documents of the product manufactured by the company. The requirements of the system have been documented as a software requirements specification (SRS). The SRS is divided into multiple scenes including user interfaces and other forms of output. The system is a Microsoft Windows based desktop application using databases as the data source. The software is using .NET[2] framework and is developed using Microsoft Visual Studio[3] as the integrated development environment (IDE). To implement ATDD/TDD, SpecFlow[4] and NUnit[5] have been used and integrated into the IDE. As one requirement in the system, the requirement to select a company, which has been described in Section 3.1, is used in this example.

[2] .NET is an application framework initially developed by Microsoft for Windows operating systems https://www.microsoft.com/net/
[3] Microsoft Visual Studio is an integrated development environment (IDE) for .NET framework https://www.visualstudio.com
[4] SpecFlow is an open source acceptance tests tool integrated with Microsoft Visual Studio http://specflow.org
[5] NUnit is a unit-testing framework for .NET languages http://nunit.org

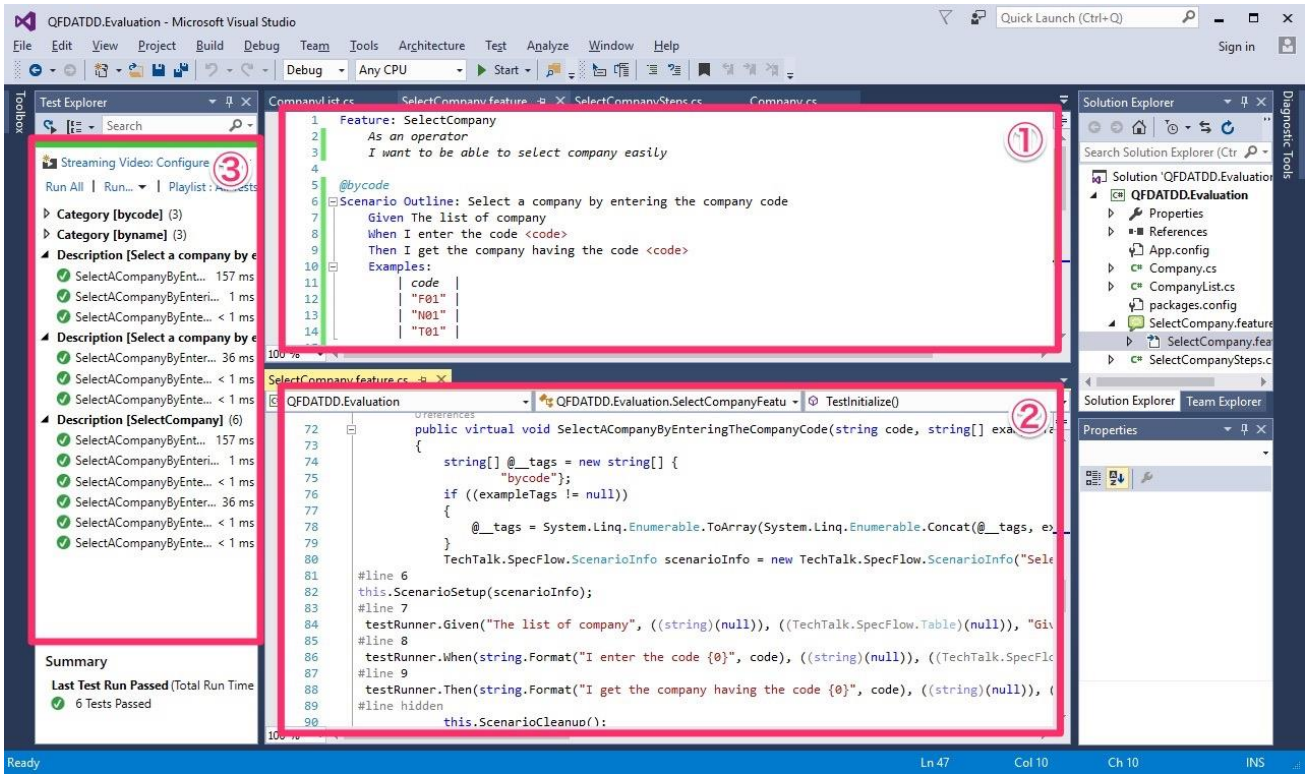

Figure 7: Example implementation of acceptance tests in the development environment

Figure 7 shows excerpts of the development environment containing the feature definition, acceptance tests codes, and the production codes. It shows how a feature of selecting a company is implemented as acceptance tests and the related test codes and production codes are being generated within the IDE.

In Figure 7, the feature definition or the user stories are shown in box number (1). The feature is written using the natural language as defined in Gherkin[6] language. Based on this feature definition, the SpecFlow automatically generates the acceptance tests which are shown in box number (2). And based on these, SpecFlow tool can be called to generate the class declaring the steps to run the acceptance tests which are not shown in the figure. Finally, after the production codes are written to make the acceptance tests pass, the acceptance tests can be executed with the result as shown in box number (3).

QFDATDD.Evaluation Test Execution Report

Generated by SpecFlow at 07/26/2017 16:55 (see http://www.specflow.org/).

Summary

Features	Success rate	Scenarios	Success	Failed	Pending	Ignored
1 features	100%	6	6	0	0	0

Feature Summary

Feature	Success rate	Scenarios	Success	Failed	Pending	Ignored
SelectCompany	100%	6	6	0	0	0

Feature Execution Details

Feature: SelectCompany

Scenario	Status	Time(s)
Select a company by entering the characters from the company name("\"Fuj\"",null) [show]	success	0.268
Select a company by entering the characters from the company name("\"NE\"",null) [show]	success	0.001
Select a company by entering the characters from the company name("\"Tos\"",null) [show]	success	0.000
Select a company by entering the company code("\"F01\"",null) [show]	success	0.025
Select a company by entering the company code("\"N01\"",null) [show]	success	0.000
Select a company by entering the company code("\"T01\"",null) [show]	success	0.000

Figure 8: Example of the acceptance tests report

[6] Gherkin is the language used to define the feature https://github.com/cucumber/cucumber/wiki/Gherkin

Furthermore, using the built-in functions from the NUnit and the SpecFlow, the test execution report can be generated like shown in Figure 8.

3.4 Summary and Discussion

As shown in Figure 9, in conventional V-model [21], the requirements are used to derive the design and their implementation. Then, after performing unit and integration tests, finally, the implementation will be validated against the requirements. The drawback is that the acceptance tests can not be performed until all codes are implemented and integrated. If the acceptance tests fail, the cost of reworking is big because the developer team has to fix the implementation, and may have to reconsider the design too, and re-runs all the tests.

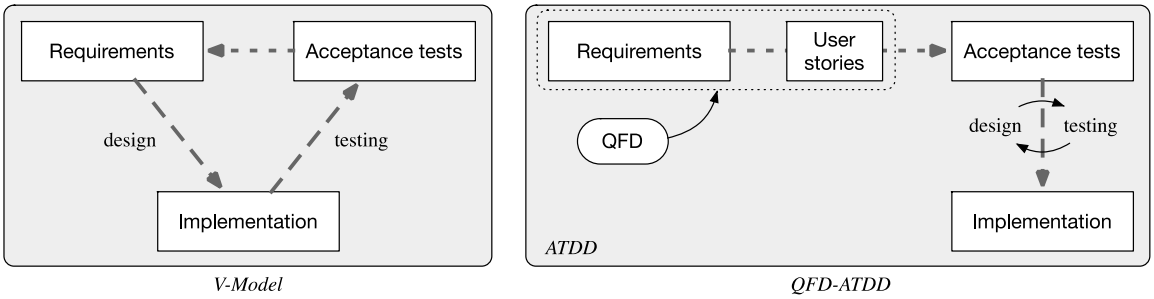

Figure 9: Comparison between conventional V-model and the QFD-ATDD model

In ATDD, the acceptances tests are first created based on user stories. Driven by these acceptance tests, design and testing are then performed simultaneously and incrementally providing rapid delivery of software functionalities. On top of that, by introducing QFD and performing its principles, the developer team, and also the customer, are work together to derive the requirements from the voices of the customer, compose the user stories, and create the acceptance tests. By performing this steps, the developer team along with the customer can be assured that the software will be developed based on the requirements and also will be validated against the requirements.

We have conducted a research regarding the implementation of TDD in the development of a business system especially in the data access layer [22]. In the research, we have successfully gained a 35% improvement of maintainability index within the codes related to data access layer developed using the TDD technique compared to those developed using the non-TDD technique. The maintainability index is a value between 0 and 100 that represents the relative ease of maintaining the code where the higher value means better maintainability [23]. The ATDD technique is built based on TDD concept. Although we have not yet conducted any quantitative analysis on the QFD-ATDD method, considering that both ATDD and TDD are based on the same concept, from the maintainability aspects, alone, we also expect the same improvement in the code maintainability with the proposed method.

The first half part of the QFD-ATDD is adapted from QFD. In this part, the requirements are derived from the voices of customer. These requirements are then deployed into user stories where one user story may contain several scenarios. These user stories are then converted into the acceptance tests to be used to drive the design and implementation. Figure 10 shows the extended QFD to derive the acceptance tests from the voices of customers in an entity relationship diagram. The diagram shows the deployment process of the voices customer into the customer requirements, subsequently into the quality table, and finally into the acceptance tests, as well as their relationships considering each of them as an entity.

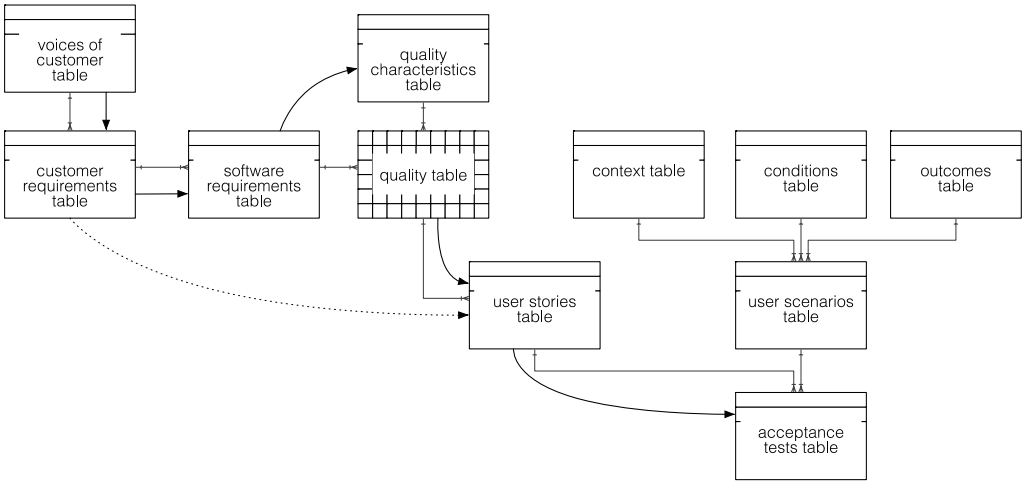

Figure 10: The extended QFD to derive the acceptance tests

Herzwurm et al. also stated that QFD can be adapted to changing requirements and/or product characteristics by proceeding QFD in many iterations which they called it the Continuous QFD (CQFD) [14]. In CQFD, customer feedback can be gathered in order to influence and possibly direct the further development. ATDD provides quick and rapid feedback of the progress of the development by acceptance tests. By quickly and rapidly receiving the feedback, the customer can then give any necessary feedback to the development to finer-grain their requirements and/or the implementation. The proposed QFD-ATDD method can address the issue of changing requirements in the more direct collaboration between the customer and the developer team.

This paper primarily discussed a method to link the user or the customer requirements to their implementation through the application of QFD and ATDD. To provide more finer-grained of the implementation, the identification of why the requirement is given should also be considered and integrated into the approach.

4. CONCLUDING REMARKS

This paper describes our idea of collaborating QFD and ATDD. QFD has been recognized as a design approach for problem-solving including the intangible and complex product like software. Applying the principles of QFD to capture the customer requirements and deploy them into functional and non-functional requirements, this paper argues that those results can be mapped into user stories which then become the acceptance tests. Using the appropriate tools, the development can then be performed based on those acceptance tests using the ATDD techniques. Considering that, the production codes can later be validated against the customer requirements rather than against the interpretation of the requirements by the developer team.

Future work includes the organization of all information covering from the initial customer needs and requirements to the user stories as well as the acceptance tests and their implementation in the productions codes. Using the relational database to organize that information will enable every member of the development project, including the customer, to view the overall product specification and implementation.

REFERENCES

[1] Didar Zowghi and Nur Nurmuliani, "A Study of the Impact of Requirements Volatility on Software Project Performance," in *Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02)*, pp. 3-11, 2002.

[2] Bill Curtis, Herb Krasner, and Neil Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268-1287, 1988.

[3] Nur Nurmuliani, Didar Zowghi, and Sue Fowell, "Analysis of Requirements Volatility During Software Development Life Cycle," in *Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04)*, pp. 28-37, 2004.

[4] Kent Beck, *Test-Driven Development by Example*.: Addison-Wesley, 2008.

[5] Stephan Wels, "Test Driven Development," in *Proceedings of Agile Seminar 2012*, August 2012. [Online]. https://sewiki.iai.uni-bonn.de/teaching/labs/xp/2012b/seminar/start

[6] Markus Gärtnur, *ATDD by Example: A Practical Guide to Acceptance Test-Driven Development*.: Addison-Wesley, 2012.

[7] Roger S. Pressman, *Software Engineering: A Practioner's Approach*, 7th ed.: McGraww-Hill, 2010.

[8] Dan North, "Behavior Modification," *Better Software Magazine*, March 2006.

[9] Sari Kroizer, "Acceptance Tests Driving Development in Scrum," in *Proceedings of the 28th Pacific Northwest Software Quality Conference (PNSQC 2010)*, pp. 263-276, 2010.

[10] Alexandre Melo Braga, Daniela Castilho Schwab, and André Luiz Vannucci, "The Use of Acceptance Test-Driven Development in the Construction of Cryptographic Software," in *Proceedings of the 9th International Conference on Emerging Security Information, Systems and Technologies (SECUREWARE 2015)*, pp. 55-60, 2015.

[11] Tuyet-Lan Tran and Joseph S. Sherif, "Quality Function Deployment (QFD): An Effective Technique for Requirements Acquisition and Reuse," in *Proceedings of 2nd IEEE International Software Engineering Standard Symposium (ISESS'95)*, pp. 191-200, 1995.

[12] Yoshimichi Watanabe, Yuichi Kawakami, and Naofumi Iizawa, "Software Requirements Analysis Method using QFD," in *Proceedings of the 18th International Symposium on Quality Function Deployment (ISQFD)*, 2012.

[13] Yunarso Anang, Masakazu Takahashi, and Yoshimichi Watanabe, "A Method for Software Requirement Volatility Analysis Using QFD," *Complex Systems Informatics and Modeling Quarterly (CSIMQ)*, no. 8, pp. 1-14, 2016.

[14] Georg Herzwurm, Sixten Schockert, and Wolfram Pietsch, "QFD for Customer-Focused Requirements Engineering," in *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pp. 330-338, 2003.

[15] Hirotaka Takeuchi and Ikujiro Nonaka, "The New New Product Development Game," *Harvard Business Report*, January 1986.

[16] Ken Schwaber and Jeff Sutherland. (2016) The Scrum Guide. [Online]. http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf

[17] Nandini Nayar, Tanu Sharma, Sushil Bansal, and Sapna Saxena, "Implementation of "XP-QFD" in a Small Scale Project," *International Journal of Computer Applications*, vol. 82, no. 10, pp. 20-24, 2013.

[18] Tadashi Ohfuji, "Tool of Quality Function Deployment (in Japanese)," *Journal of Precision Engineering*, vol. 75, no. 11, pp. 1289-1292, 2009.

[19] Hisakazu Shindo, *Design Approach to Problem Solving: An Approach to TQM Revitalization (in Japanese)*.: JUSE Press, 2011.

[20] Tadashi Ohfuji, "Tool of Quality Function Deployment (in Japanese)," *Journal of Precision Engineering*, vol. 75, no. 11, pp. 1289-1292, 2009.

[21] Christian Bucanac, "The V-Model," University of Karlskrona/Ronneby, 1999. [Online]. http://www.bucanac.com/documents/The_V-Model.pdf

[22] Yunarso Anang, Masakazu Takahashi, and Yoshimichi Watanabe, "Implementation of Test-Driven Development in Data Access Layer within a Business System Development," in *forthcoming Proceedings of the Asian Network for Quality Congress (ANQ2017)*, 2017.

[23] Zain Naboulsi. (2011) MSDN Blogs. [Online]. https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/

BIOGRAPHIES OF THE AUTHORS

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

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, Japan from 2005 to 2008. He is presently a professor in University of Yamanashi, Japan. 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.