13th International Conference on Intelligent Software Methodologies, Tools, and Techniques (SOMET14), Langkawi, Malaysia, September 22-24, 2014, © IOS Press

Automatic Acceptance Test Case Generation From Essential Use Cases

Massila Kamalrudin[a,1], M. Nor Aiza[a], John Grundy[b], John Hosking[c], Mark Robinson[d]
[a] *Innovative Software System & Services Group, Universiti Teknikal Malaysia Melaka, Malaysia*
massila@utem.edu.my, P031320015@student.utem.edu.my
[b] *Centre for Complex Software Systems & Services, Swinburne University of Technology, PO Box 218, Hawthorn Victoria 3122, Australia*
jgrundy@swin.edu.au
[c] *College of Engineering and Computer Science, Australian National University, Canberra, ACT, Australia*
john.hosking@anu.edu.au
[d] *Fulgent Corporation, USA*
marcos@fulgentcorp.com

Abstract. Requirements validation is a crucial process to determine whether client-stakeholders' needs and expectations of a product are sufficiently correct and complete. Various requirements validation techniques have been used to evaluate the correctness and quality of requirements, but most of these techniques are tedious, expensive and time consuming. Accordingly, most project members are reluctant to invest their time and efforts in the requirements validation process. Moreover, automated tool supports that promote effective collaboration between the client-stakeholders and the engineers are still lacking. In this paper, we describe a novel approach that combines prototyping and test-based requirements techniques to improve the requirements validation process and promote better communication and collaboration between requirements engineers and client-stakeholders. To justify the potential of this prototype tool, we also present three types of evaluation conducted on the prototpye tool, which are the usability survey, 3-tool comparison analysis and expert reviews.

Keywords. Requirements Engineering, Validation, Essential Use Case (EUC), Automated Acceptance Test

Introduction

Capturing correct and consistent requirements from client-stakeholders is often difficult, time consuming and error prone [1][2]. Therefore, it is important to validate requirements at the earliest stage of software development in order to be able to detect and prevent errors in requirements specifications.

Currently, various requirements validation techniques, such as requirements review, inspections, prototyping, model-based, requirements testing and viewpoint-

[1] Massila Kamalrudin-massila@utem.edu.my.

oriented requirements validation have been applied to develop quality software [3][4][5]. Each has its own advantages and disadvantages depending on the needs of the organization. Studies have recognized two effective techniques to identify requirements defects: the requirements prototyping and test-based requirements validation. The former is beneficial as it helps users to visualize requirements by providing a prototype of the system. The prototype is also reusable in other activities, such as the system design and user interface development [4]. The test-based requirements validation helps by defining test cases to ensure each requirement is testable, providing a means to determine when a requirement is satisfied. It is also an effective way of exposing problems, such as incompleteness, inconsistency and ambiguity by suggesting possible ways of testing the requirements [3]. However, both techniques are expensive and time consuming, as they require resources and efforts to develop the prototype and write the test cases. Moreover, to our knowledge there have been very few successful attempts made to support rapid prototyping and model-based testing.

Previously, we developed a rapid prototyping support for an end-to-end user requirements validation tool [1]. We used an abstract Essential User Interface (EUI) prototype and concrete formed-based User Interface (UIs) to help stakeholders visualize and walk-through rapid prototypes based on their elicited requirements. To further improve the requirements validation process, we enhanced our tool by providing executable UIs with a set of related abstract test cases for testing the requirements. The abstract test cases are derived from the developed EUC and EUI patterns. These executable UIs also allow stakeholders to experiment with generated prototypes from their requirements. We conducted a small user study to evaluate the usability of our prototype tool. An analysis of the results of this preliminary evaluation showed that our approach could improve the validation process and promote better communication and collaboration between Requirements Engineers (REs) and client-stakeholders.

1. Related Works

Extensive research has been conducted in requirements validation especially on model and test-based validation. Test-based requirements, or requirements testing, is one of the most commonly used techniques in requirement validation. This involves the acceptance testing, where users or developers create test cases to validate requirements. In our preliminary study, we conducted a comparison analysis of a few selected automated acceptance-testing tools from the existing literature and experience reports [6]. We found that automated acceptance tools are beneficial for both business stakeholders and the development team to reduce the time and efforts in the manual acceptance testing process.

In a different study done by Escalona et al. [7], they presented a rigorous comparative analysis of 13 different characterized approaches on functional test cases generation from functional requirements. They claimed that development teams need to have a good understanding of most of the approaches involving generating functional test cases from models, such as the UML activity diagrams. They also found that some of the approaches are highly systemized, though they do not offer a tool support. Overall, they concluded that there is no definitive approach that effectively generates functional test case automatically.

The UML diagrams are the most commonly used modeling technique in model-based requirements validation. For instance, EuRailCheck [8] used a set of UML concepts and diagrams, such as classes, class diagrams, state machines, and sequence diagrams to formalize the categorized requirement fragments of a requirements document. The authors claimed to support an end-to-end methodology for the analysis of requirements and guarantee traceability, aspects that received positive feedbacks in validating system for different railway organizations. However, the tool requires human intervention in all of the three main functions of the tool: fragmentation, categorization and formalization of the requirements.

B. Hasling, H. Goetz and K. Beetz [9] have used UML use case models for creating system test cases to ensure the testability of requirements. The supporting tool i.e. TDE/UML allows efficient use of this approach and assist for creation of system tests. It has features to prune the number of test paths through the model. However, it still needs some sort of prioritization scheme to prioritize the generated test for which need to be executed first.

Z. Bin and W. Anbao [10] have proposed a method by integrating use case models and task model to generate functional and user interface test cases. They used the integration of both models with the formal semantic of finite state machine (FSM) to generate more complete and detailed test cases. This approach is heavyweight which contrast to our work to be easier as it is automated and uses semi-formalise model and prototype to visualise the output.

The UCAT (Use Case Acceptance Tester) [11] provides an automated support for executing acceptance tests. However, this tool requires end users to possess substantial expertise in modeling use cases. The efficiency of the developed acceptance test is highly dependent on the quality of the use case model, which is heavily reliable on the skill and experience of the Requirements Engineer.

R. Ibrahim et al. [12] proposed a tool called GenTCase to generate test case automatically according to the system's requirements. The test cases can be used as a checklist for a programmer to validate that the system meets its requirements. The tool is found to be able to reduce the cost for system testing and save time of producing the test case manually. In similarity, this tool also concern on capturing the functional requirements of the system only, for which the non-functional requirement need to captured and tested outside the tool. Instead of using natural language for the system requirements, this tool requires the user to place the use case diagram, flow of event and sequence diagram, where the consistency of test cases generated are highly dependant on the consistency of the flow of events and sequence diagrams.

The Essential Use Cases (EUCs), a semi-formal model for requirements specification, have been used in MaramaAI for capturing and validating business requirements [1]. The EUCs are automatically extracted from natural language requirements and translated into a low-fidelity "Essential User Interface" prototype. Although this tool is useful for consistency management and requirement validation, it does not support user acceptance testing.

2. Our Approach

We have been exploring a new approach that combines a rapid UI prototyping and requirements-based testing to validate user requirements using a black-box testing strategy [13][14]. Figure 1 presents an overview of our approach. The process starts by

capturing the client-stakeholder requirements in the form of user story or use case narrative (1). These requirements are analyzed to generate an EUC model (2). A low-fidelity EUI prototype is then derived from the EUC model (3). This model is transformed to a concrete HTML form-based UI (4). The next step is the user acceptance testing to validate the requirements. A set of abstract test cases and associated executable UIs that match the EUI prototype are generated to guide this testing (5). Users can indicate whether the tests pass or fail by choosing a radio button option. This tool aims to help validating user's requirements as well as facilitating effective communication that promotes collaboration among the client-stakeholders. The generated test cases can also serve as formal documentations and they are reusable in the final testing phase.

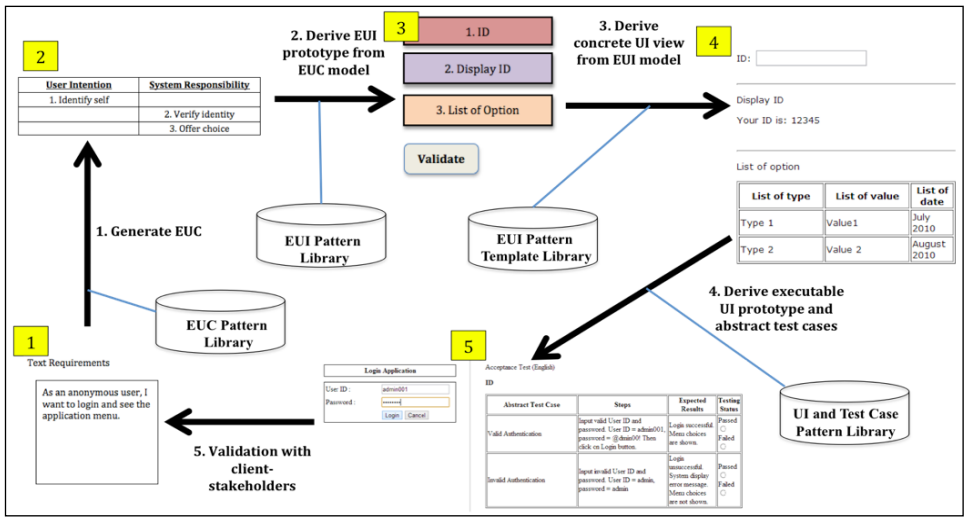


Figure 1. An Overview of Our Approach

3. Usage Example

Figure 2 and 3 illustrate a usage example of our prototype tool using the requirement: *As an anonymous user, I want to login and see the application menu.*

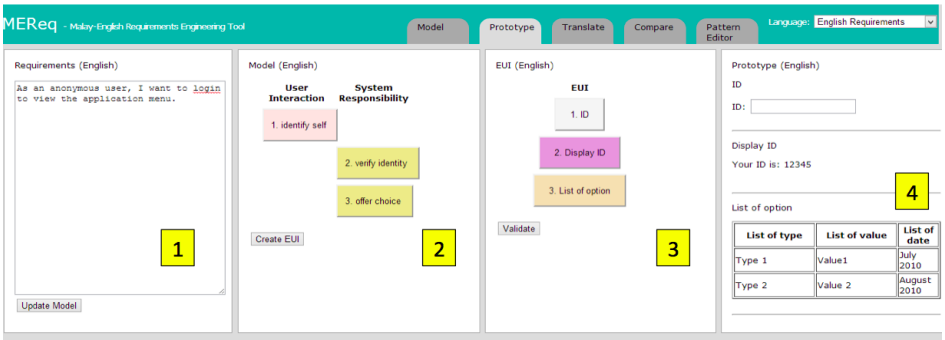


Figure 2. UI view of MEReq in use

In Figure 2, from the requirements text (1), the EUC models are derived (2) and mapped to a low-fidelity EUI model (3). This EUI prototype is then mapped to the concrete UI (4). Figure 3 illustrates the view of the associated executable UI, labeled as (A) and its test cases generated based on the EUC and EUI models. The generated test cases contain the description of the test, steps and expected results. Users may test the UI by providing the input data given in the test case and view the expected results. Figure 3 shows the UI view of the testing result (B), which is displayed when users run the "Valid Authentication" test case. Then, users may indicate the status of the testing in the "Testing Status" column. Test results are saved in the database and can be retrieved for future reference.

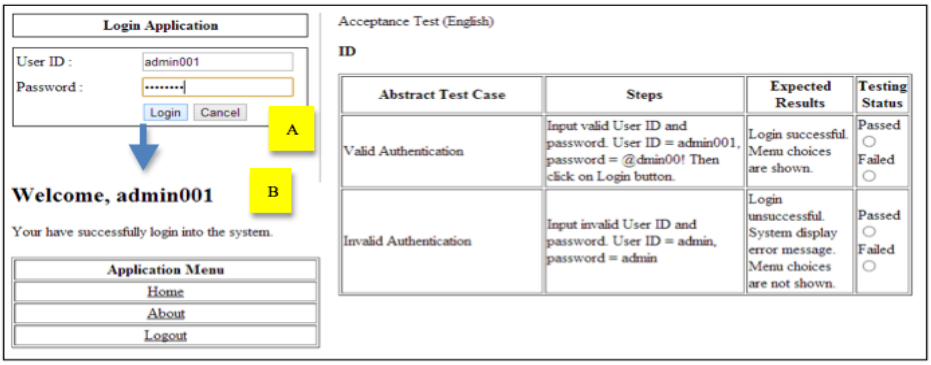


Figure 3. The UI view of the executable UI and its related test case

4. Implementation

We have enhanced our MEReq [15] tool to map the EUI prototypes and concrete UIs to an executable UI and related test cases. Figure 4 outlines the high level architecture of our prototype tool, the TestMEReq. A user uses the tool via a web browser or tablet device. The tool UI contains three key elements: the textual natural language requirements, corresponding EUC and EUI models (1). An Apache Web server hosts a Java Server Faces implementation of this web interface (2). A MySQL database server contains the EUC and EUI pattern libraries along with the EUI template and abstract test case pattern libraries (3).

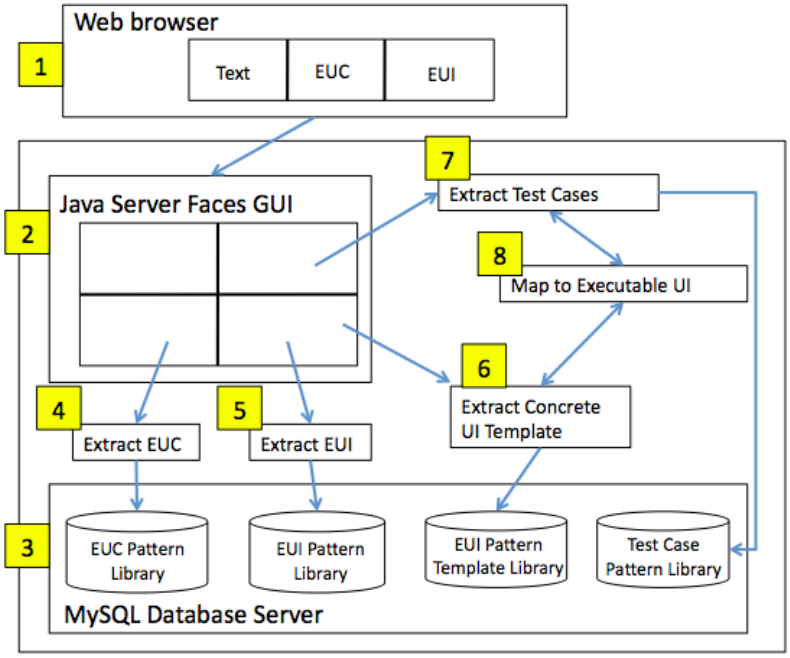


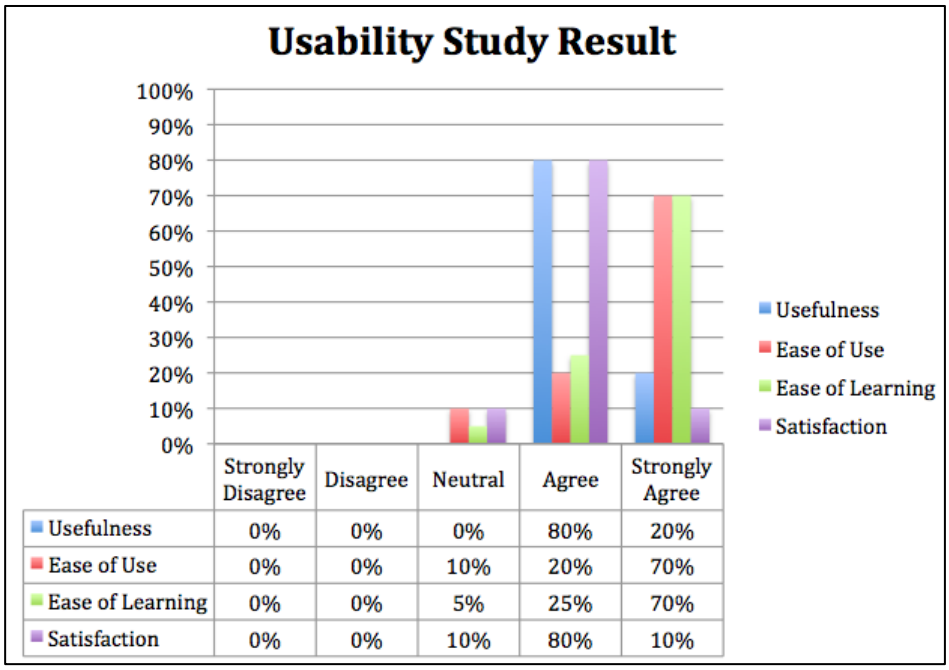
Figure 4. High-Level Architecture of MEReq.

The former, MEReq [15] supports the extraction of EUCs from the requirement text (4). To do this, we parse the requirements text to locate an essential interaction phrase in the text to match a library pattern, and then we use this to identify the associated abstract interaction. From here, the EUC model is generated (4). Then, the associated EUI model together with the concrete UI templates are extracted based on the generated EUC model (5). The EUC abstract interaction patterns are enhanced to include the definition, preconditions and post-conditions of the EUC and EUI models. Here, the black-box testing strategy is applied to derive the abstract test cases that are predefined in the pattern library. A Java-implemented module is developed to automatically parse the EUC to generate the abstract test cases (7). Then, the generated concrete UI and abstract test cases are mapped to the associated executable UIs (8).

4.1. Usability Evaluation

We conducted a preliminary evaluation on our prototype tool to evaluate its usability. 20 postgraduate students majoring in Software Engineering and Artificial Intelligent were recruited to carry out the evaluation. These students have sufficient understanding of the concept and methodology of essential use cases. Each participant was given a brief description and tutorial on how to use the tool. Then, they were asked to explore the tool with the same requirements presented in Section 4. Although this requirements scenario is simple, it is realistic and represents the common functions in most software application. Finally, they were asked to complete a survey questionnaire. This survey aimed at gaining insights of the usability of this protoype tool with respect to its usefulness, ease of use, ease of learning and user's satisfaction. The survey consisted of

three questions for each question block and was measured using five level Likert scale. The results of the usability study are shown in Figure 5.



	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Usefulness	0%	0%	0%	80%	20%
Ease of Use	0%	0%	10%	20%	70%
Ease of Learning	0%	0%	5%	25%	70%
Satisfaction	0%	0%	10%	80%	10%

Figure 5. Usability Study Results

In term of the usefulness of the tool, 80% of the participants agreed and 20% strongly agreed that the tool was useful for validating requirements. They also found that the tool to be very easy (70%) and always easy (20%) to use. In term of the ease of learning, 70% of the participants claimed that it is very easy to learn since the flow and the interface design of the tool are simple and user-friendly. Additionally, 80% of the participants were very satisfied, and 10% of the participants were always satisfied with the tool for requirements validation as it allows them to visualize and walk-through the requirements. The abstract test cases and workable UIs help them to visualize and understand the high-level description of their requirements. Overall, the usability results show that our prototype tool is useful, easily used and easily learnt. Users also exert high satisfaction when using it. However, we believe that there are still rooms for improvement on this prototype tool.

4.2. Tools Comparison

We also reviewed and compared our prototype tool with FitNesse [16] and Selenium [17]. Our goal was to evaluate the usage of the three tools in acceptance testing. We examined how they implement or create the test cases. We also evaluated their test case readability for non-technical users. Finally, we evaluated their usability in terms of user-friendliness, ease of use and learnability. Table 1 presents a summary of our findings.

Table 1. Summary of Tools Comparison

Tools Name		TestMEReq	FitNesse	Selenium
Requirements Representation	Formal	-	-	-
	Semi-Formal	√	-	-
	Informal (NL, User Stories, Table)	√	√	√
Test Implementation	Auto-generated	√	-	-
	Coding	-	√	-
	Capture & Replay	-	-	√
Readability	Test case Readability	√	√	-
Usability	Ease of use & learnability	√	√	√
	User friendly	√	√	√

FitNesse is an open source tool built on top of Fit, which is used for automating acceptance test cases. It is an easy tool as it uses wiki web server, requiring a minimal learning curve. It is a collaborative tool that aims to improve communication between customer, analyst, and developer. It allows users to upload requirements and related Fit tables containing inputs and expected outputs of tests [16][18].

Selenium is an open source tool for simple functional testing of a web application. Tests are written as simple HTML tables [19]. However, it may not be easy for non-technical users to read and write tests. Whilst FitNesse requires the developer to write the code (Fixture) to link the test cases with the system under test, Selenium has record and replay features to build test scripts. These features record users' actions and export them as reusable scripts that can be later executed [17][20][21].

In comparison, our TestMEReq support semi-formal requirements representations: EUC and EUI model and in informal requirements in a form of user stories and scenarios. At the same time, an workable UI is also generated where users can test and run the UI by providing the input data given in the test cases. These test cases are automatically generated from our test case pattern library, instead of writing the fixture code as FitNesse or building the code using Capture and Replay feature as Selenium. Our test cases are readable and easy to understand even by the non-technical users. Based on the evaluation, it is found that our prototype tool is easy to use and learn and also user friendly.

4.3. Expert Review

For further evaluation, we also conducted two interviews with the experts in the field of software development and testing. The first interview was conducted with a project manager in Fulgent Corp, USA. Our interview session was conducted through emails and Skype. Based on the interview, he agreed that the tool is easy to use and learn. However, he highlighted the scalability issue of the tool that may arise when handling larger requirements. For improvement, he suggested to include a traceability link

between functional requirements and abstract test cases that are currently not supported for large requirements.

The second expert review was derived from a face-to-face interview session with a project manager in IBM, Malaysia, who manages a testing team in IBM. Based on the interview, he agreed that this tool can facilitate validating and clarifying requirements through workable UIs and abstract test cases. He commented that although this tool looks similar to some existing tools, it is so much simpler and easier to use. He suggested to add a function of generating a report on testing results in the form of graph representation for added value to the industry users like IBM.

4.4. Limitations

Our prototype tool has two main limitations. Firstly, this tool cannot generate test cases that are not defined by the EUC and EUI abstract interaction patterns. It requires further enhancement on the pattern editor to allow new test cases to be updated or created based on the EUC and EUI patterns. Secondly, it lacks the flexibility to allow users to upload developed test scripts. We believe that these issues can be solved if we integrate our tool with other existing testing tools, such as the FitNesse or Selenium to better support the generation of test cases/scripts.

5. Impact And Future Directions

Often in software engineering efforts, it is a major challenge to elicit correct, consistent, and complete requirements from all client-stakeholders. There are numerous requirements validation techniques, but most of them are tedious, expensive, and time consuming. Our approach combines rapid prototyping and acceptance testing to validate user requirements. A preliminary evaluation of our approach suggests that our tool facilitates the requirements validation process and promotes better communication and collaboration among client-stakeholders.

For future work, we intend to conduct an industrial evaluation with the Fulgent Corp to corroborate our preliminary findings. We also plan to integrate our tool with existing automated testing tools, such as the FitNesse and Selenium. The former will allow users to write their own test cases to upload into our database. The integration of Selenium will allow replay-based test case/scripts generation from our executable UIs.

6. Acknowledgements

We would like to thank Fulgent Corporation, USA and FRGS grant: FRGS/2/2013/ICT01/FTMK/02/2/F00185 for funding this research. We also would like to thank UTeM and Dr. Safiah Sidek for her assistance.

References

[1] M. Kamalrudin and J. Grundy, "Generating Essential User Interface Prototypes to Validate Requirements," in *IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 10–13.

[2] M. Kamalrudin, J. Grundy, and J. Hosking, "Tool Support for Essential Use Cases to Better Capture Software Requirements," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 255–264.

[3] S. B. Saqi and S. Ahmed, "Requirements Validation Techniques practiced in industry : Studies of six companies," Blekinge Institute of Technology, Sweden, 2008.

[4] U. A. Raja, "Empirical studies of requirements validation techniques," in *2nd International Conference on Computer, Control and Communication*, 2009, pp. 1–9.

[5] F. Yousuf, Z. Zaman, and N. Ikram, "Requirements Validation Techniques in GSD : A Survey," in *IEEE International Multitopic Conference*, 2008, pp. 553–557.

[6] M. Kamalrudin, S. Sidek, M. Nor Aiza, and M. Robinson, "Automated Acceptance Testing Tools Evaluation in Agile Software Development," *Sci. Int.*, no. 4, pp. 1053–1058, 2013.

[7] M. J. Escalona, J. J. Gutierrez, M. Mejías, G. Aragón, I. Ramos, J. Torres, and F. J. Domínguez, "An overview on test generation from functional requirements," *J. Syst. Softw.*, vol. 84, no. 8, pp. 1379–1393, 2011.

[8] R. Cavada, A. Cimatti, A. Mariotti, C. Mattarei, A. Micheli, S. Mover, M. Pensallorto, M. Roveri, A. Susi, S. Tonetta, and F. B. Kessler, "Supporting Requirements Validation : the EuRailCheck tool," in *IEEE/ACM International Conference on Automated Software Engineering*, 2009, pp. 665–667.

[9] B. Hasling, H. Goetz, and K. Beetz, "Model Based Testing of System Requirements using UML Use Case Models," in *International conference on Software Testing, Verification, and Validation*, 2008, pp. 367–376.

[10] Z. Bin and W. Anbao, "Functional and User Interface Model for Generating Test Cases," in *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, 2012, pp. 605–610.

[11] M. El-Attar and J. Miller, "Developing comprehensive acceptance tests from use cases and robustness diagrams," *Requir. Eng.*, vol. 15, pp. 285–306, 2010.

[12] R. Ibrahim, M. Z. Saringat, N. Ibrahim, and N. Ismail, "An Automatic Tool for Generating Test Cases from the System's Requirements," in *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, 2007, pp. 861–866.

[13] T. Hammel, R. Gold, and T. Snyder, *Test-Driven Development: A J2EE Example*. New York, USA: Apress, 2005.

[14] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The Art of Software Testing*, 2nd Editio. Hoboken, New Jersey: John Wiley & Sons, Inc, 2004.

[15] M. Kamalrudin, S. Sidek, and N. Yusop, "MEReq: A Tool to Capture and Validate Multi-Lingual," in *The 13th International Conference on Intelligent Software Methodologies, Tools and Techniques*, 2014.

[16] "FitNesse." [Online]. Available: http://fitnesse.org/. [Accessed: 03-Jul-2013].

[17] Selenium Project, "SeleniumHQ," 2013. [Online]. Available: http://www.seleniumhq.org.

[18] B. Haugset and G. K. Hanssen, "The Home Ground of Automated Acceptance Testing: Mature Use of FitNesse," in *2011 AGILE Conference*, 2011, pp. 97–106.

[19] A. Holmes and M. Kellogg, "Automating Functional Tests Using Selenium," in *Proceedings of Agile 2006 (Agile'06)*, 2006, pp. 270–275.

[20] R. A. Razak and F. R. Fahrurazi, "Agile testing with Selenium," in *2011 Malaysian Conference in Software Engineering*, 2011, pp. 217–219.

[21] Selenium Project, "Selenium Documentation." pp. 1–158, 2010.