3rd International Workshop on Software Engineering Research and Industrial Practice (SER&IP), May 2016.

TestMEReq: Generating Abstract Tests for Requirements Validation

Nor Aiza Moketar, Massila Kamalrudin, Safiah Sidek
Innovative Software System and Services Group,
Universiti Teknikal Malaysia Melaka,
Melaka, Malaysia
nor.aiza09@gmail.com, {massila,safiahsidek}@utem.edu.my

Mark Robinson
Fulgent Corporation,
USA
marcos@fulgentcorp.com

John Grundy
School of Information Technology
Deakin University
Melbourne, Australia
j.grundy@deakin.edu.au

ABSTRACT

This paper introduces TestMEReq, an automated tool for early validation of requirements. TestMEReq supports requirements engineers (REs) in the validation of the correctness, completeness and consistency of elicited requirements with minimum effort and time through generated abstract tests components: test requirements and test cases, and a mock-up prototype of the user interface (UI). Abstract tests are derived from abstract models called Essential Use Cases (EUCs) and the Essential User Interface (EUI). Our evaluation results show that TestMEReq is useful in the requirements validation process: it reduces the effort and time spent to ensure good quality requirements.

CCS Concepts

• Software and its engineering□Software Creation and Management□Designing Software□Requirements Analysis • Software and its engineering□Software Creation and Management□Software verification and validation□Process validation□Acceptance testing

Keywords

Abstract test, Essential Use Cases, Essential User Interface, requirement-based testing, requirements validation

1. INTRODUCTION

Requirements are the main source of knowledge for software development and they are often elicited and specified in natural language (NL). However, NL requirements are error-prone due to misunderstanding, miscommunication and misinterpretation during elicitation and negotiation [1][2]. Imprecise requirements are very risky to software project development [2][3] as they may lead to incorrect implementation of software that does not meet the needs and expectations of users. These requirements defects can also result in time and cost overruns to rectify the software at a later stage, and can eventually lead to the failure of a software project. Therefore, it is important to properly validate requirements at an early stage in the development process. The purpose of requirements validation is to confirm that the requirements are correct, complete, consistent and agreed by client-stakeholders prior to the implementation of the software project [4][5].

For this, requirements testing and prototyping have been found to be useful techniques to validate requirements. The technique of requirements testing defines test cases from the requirements. This allows REs to ensure that each requirement is testable, thus providing a means to determine when a requirement is satisfied. By suggesting possible tests for requirements, this technique is an effective way to detect problems such as incompleteness, inconsistency and ambiguity [4]. Furthermore, the generated test cases are reusable during testing activities [4][5]. The use of a requirements prototype helps REs to detect defects by visualising the realised requirements. The prototype is also reusable in other activities, such as system design and user interface development [4][5]. Although these two techniques are capable of detecting defects, they are expensive and time consuming.

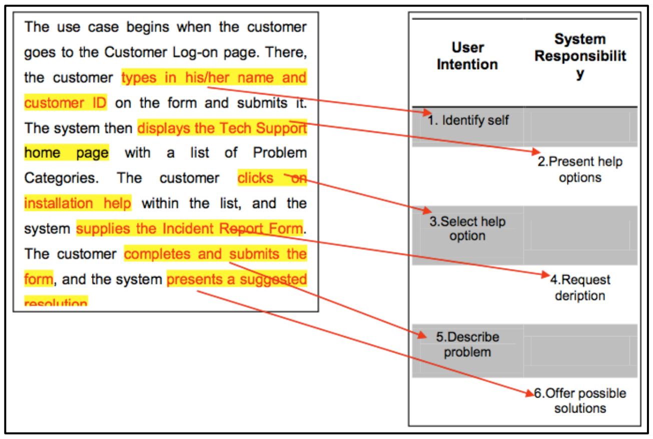
Motivated by the usefulness and the expense of requirements testing and prototyping, we have created TestMEReq, an automated requirements validation tool that is able to automatically generate a combination of abstract test cases and mock-up UI prototypes from semi-formalised EUC and EUI models. Our automated approach assists requirements engineers in validation of requirements with stakeholders, and helps to reduce the expense of generating and designing test cases and UI prototypes.

2. BACKGROUND

We use the term ‘abstract tests’ to refer to our test requirements and test cases that are generated from the semi-formalised abstract model called the Essential Use Cases (EUC) and Essential User Interface (EUI) model. An abstract test is a high-level test requirement and test case that represents a requirements scenario. In contrast to concrete tests, an abstract test does not contain any details of the test environment, test protocol, or configuration for the test component.

An EUC is a structured narrative, expressed in the language of the application domain and the users. It is composed of a simplified, abstract, technology-free and implementation-independent description of a single task or interaction [6][7]. An EUC is a complete, meaningful, and well-designed interaction from the point-of-view of users. It represents a particular role in relation to a system and embodies the purpose or intentions underlying the interaction. EUCs enable users to ask fundamental questions such as "what's really going on" and "what do we really need to do" without letting implementation decisions get in the way. These questions often lead to critical realisations that allow users to rethink, or reengineer the aspects of the overall business process. Figure 1 shows an example of natural language requirements (left hand side) and an example of an EUC (right hand side) while capturing the requirements (adapted from [7]). The natural language requirements from which the important phrases are extracted (highlighted) are shown on the left hand side of Figure 1. From the natural language requirements, a specific key phrase (essential requirement) is abstracted and is shown in the EUC on the right hand side of Figure 1. As shown in Figure 1, the EUC depicts two interrelated sets of information: the user intentions and system responsibility.

An EUI prototype is a type of abstract prototype or paper prototype that is a low-fidelity model. Also known as a "UI prototype" for a software system, it represents the general ideas rather than the exact details of the UI [7][8]. An EUI prototype represents the user interface requirements in a technology independent manner; just as the EUC models do for the behavioural requirements. An EUI prototype is particularly effective during the initial stages of user interface prototyping for a system. It models user interface requirements that are evolved through analysis and design to the final user interface of a system [8]. It also allows some exploration of the usability aspects of a system. Figure 2 shows an example of an EUI prototype developed from EUC model. The possible UI functionality at a high level of abstraction is captured from the user intention/system responsibility dialogues.


The use case begins when the customer goes to the Customer Log-on page. There, the customer types in his/her name and customer ID on the form and submits it. The system then displays the Tech Support home page with a list of Problem Categories. The customer clicks on installation help within the list, and the system supplies the Incident Report Form. The customer completes and submits the form, and the system presents a suggested resolution

User Intention	System Responsibility
1. Identify self	
	2.Present help options
3.Select help option	
	4.Request deription
5.Describe problem	
	6.Offer possible solutions

Figure 1. Example of a natural language requirement (left hand side) and example of EUC (right hand side).

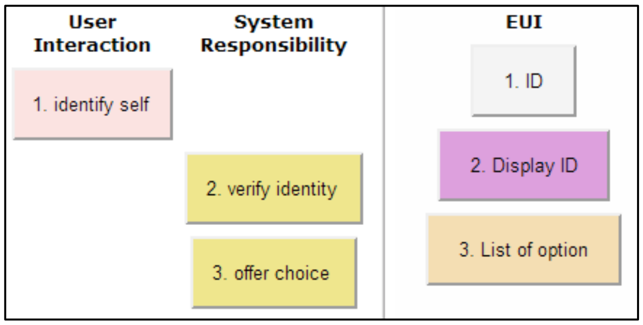


Figure 2. Example of a EUI prototype from the EUC model.

Both EUC and EUI play important roles in our work. The EUC provides a simpler and shorter form of dialogue between the user and the system compared to the conventional use case. This dialogue provides the key information of the input and output (expected results) for our test cases. An interaction (input and output) between the user and the system can generate one or more test requirements. This dialogue also provides information for the test procedures/steps in our test cases. The EUI prototype model provides a guide for the important elements to be included in our mock-up UI prototype. These two models are crucial to ensure the correctness, completeness and consistency of generated abstract tests and mock-up UI prototypes for users' requirements.

3. OUR TOOL: TESTMEREQ

We have developed an automated support tool, called TestMEReq, to assist requirements engineers (REs) in validation of requirements captured from the client-stakeholders. Our tool integrates the abstract models: EUC and EUI, with requirements-based testing and rapid prototyping techniques. TestMEReq aims to assist in an early requirements validation process by automatically generating abstract tests and a mock-up UI prototype from EUC and EUI models. This work is an extension of our previous work [9][10][11]. Previously we have described our approach and automated tool called MEReq to support translation of natural language (NL) requirements into semi-formal abstract interaction and EUC models. For this we have developed two supporting libraries: Essential Interactions and EUC Interaction pattern library. In this new work, we adopt the same approach to develop new pattern libraries: test requirements and test cases pattern libraries to help in automatic generation of abstract tests from EUC model. This automatic approach could help to lessen human intervention in writing tests and make it possible to detect defects at the initial stage of software development. In addition, prevention action can also be planned, which will lead to cost-effective software development.

Figure 3 presents an overview of our approach. To employ our approach, we suggest that every client-stakeholders get involved in the requirements validation process, especially between client/end user, requirements engineer, domain experts, developer and tester. Implementing the RBT methodology, our approach is divided into three main validation processes: initial ambiguity and requirements models (EUC and EUI) review [A], abstract tests consistency review [B] and design and codes review [C].

The first stage of validation is the initial ambiguity and requirements models review as labeled as [A] in Figure 3. The process starts by capturing the client-stakeholder requirements as a user story or use case scenario. Next, the requirements are transformed into EUCs model using TestMEReq (1). The requirements are analysed with the EUC pattern library to generate an EUC model. A low-fidelity EUI prototype is then derived from the EUC model using the EUI pattern library (2). Here, the domain expert can perform initial ambiguity reviews by checking the correctness and completeness of the generated EUC and EUI models in relation to the textual requirements.

In the second stage of validation (labelled as [B]), a set of test requirements are generated from the EUC model (3). In order to accomplish the automatic inference of test requirements, we have created a test requirements pattern library. We have set the test requirements syntax for development of the pattern library. This is to ensure consistency and uniformity of our test requirements statements. Next, a set of test cases is derived from the test requirements using our test case pattern library (4). The requirements authors, domain experts, software developer and tester may review and validate the generated test requirements and test cases. They can rectify the requirements associated with the test cases of any error found in the generated test cases. At the same time, the test cases can be corrected and redesigned. This process can also help developers to gain a better insight of the software to be developed by reviewing the test cases.

In the final stage of validation, design and code review (labelled as [C]), the associated mock-up UI prototype is generated from the test requirements (5). To do this, a set of defined test scripts to execute the test cases are developed and stored in the library. The client-stakeholder may use the test cases in the design review to validate the requirements and determine if the UI prototype meets the requirements. The test cases also can be used to validate that each code module (test scripts) delivers what is expected. Currently the test scripts are not accessible to the user/develop to review. However, the client-stakeholder may experiment with the mock-up UI prototype by referring to the generated test cases and comparing the actual behaviour with the expected behaviour of the requirements. At this point, the client-stakeholder may validate and confirm their requirements by indicating a testing result. If the client-stakeholder accepts the executed test cases, it indicates that 100 per cent of the requirements have been verified and validated, providing a higher level of confidence to make decisions during the design and development phase.

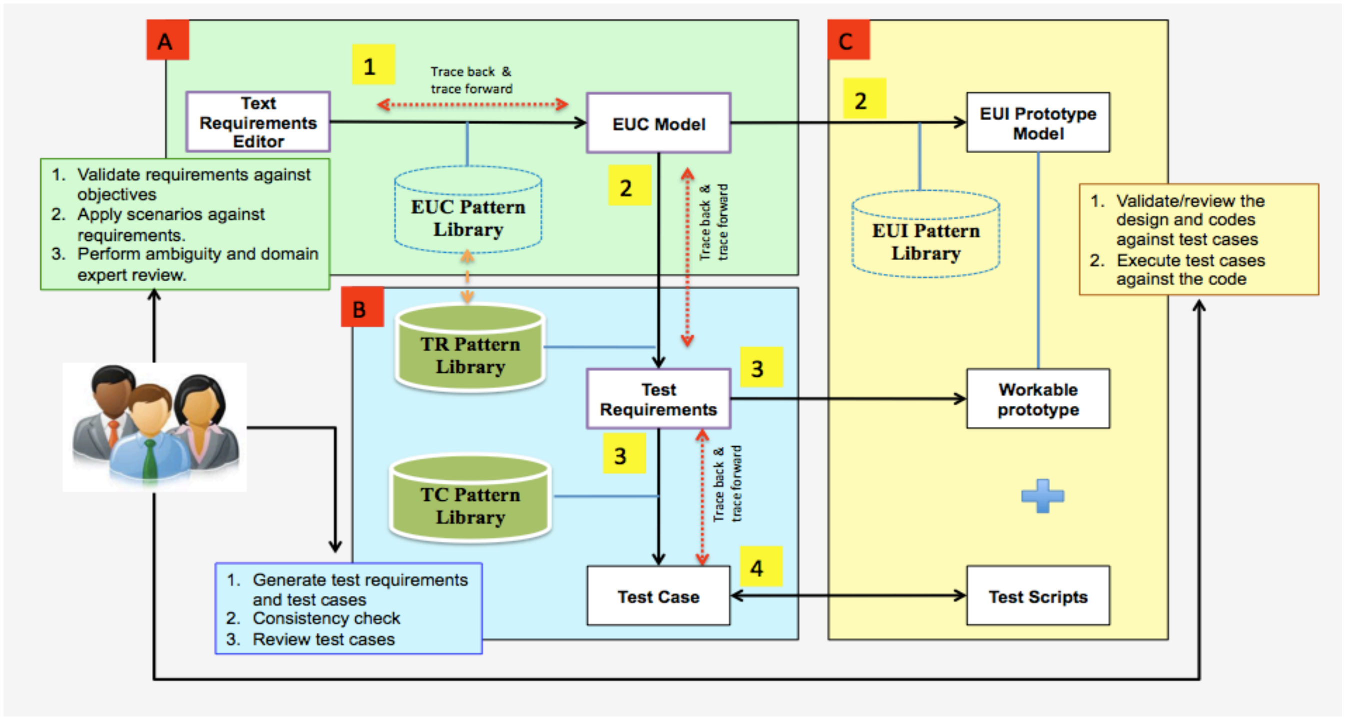
We also have embedded a traceability function to allow the user to trace back and forth between the textual requirements, the EUC and EUI models, the test requirements, and the test cases to ensure their correctness, completeness and consistency. Figure 7 illustrates the use of the traceability function from test case to textual requirements. When a test case's ID is clicked, our tool will highlight the related test requirements (in yellow). In this example, the test case ID "001" is mapped to the first test requirement. The test requirement is mapped to the EUC model "Identify Self", which is linked to the keyword "login" from the textual requirements. The RE also can trace-back from the EUI model to the textual requirements. Figure 7 shows that the EUI of "ID" is mapped to the EUC of "Identify Self" and to the same keyword in the textual requirements.

3.1 Test Requirements and Test Cases Pattern Libraries

We have developed test requirements and test cases pattern libraries to support generating our abstract tests. Previously, we collected and categorized phrases from various types of natural language requirements and stored them as essential interaction. These phrases are stored in our EUC pattern libraries [10]. To date, we have expanded the library by adding approximately 400 new phrases from various requirements domains such as healthcare, car rental, payroll system, purchasing order system and e-Learning. From here we come up with about 120 patterns of abstract interaction. For this, we have approximately 208 abstract interaction (EUC) patterns in total. We further enhanced this pattern library and categorised the abstract interaction into two categories: "Input" and "Output". The "Input" abstract interaction represents the action of the actor (user or system). Meanwhile, the "Output" abstract interaction represents the expected output for the test case. From this input-output relationship, we have identified about 300 test requirements and 624 test cases. Our abstract test patterns have been verified and evaluated by experts that include practicing test engineers, requirements engineers and software validation and verification lecturers. Our test requirements patterns were written in "action" verbs and words, such as "Validate that ...", "Verify that …" and "Test that …". In order to store the test requirements in the pattern library, we have defined the syntax rules for the sentence structure as the following:

<Action verbs> [Actor]<Auxiliary verbs> [Action] [Condition]

The items in square bracket are compulsory. The "actor" describes who is interacting with the system. It can be the user (student, customer, machine) or the system itself. The "action" states what is the user want/intended to do. The "condition" describes the circumstances of the action. This is to determine whether the test requirements are for positive or negative tests.



A
1
Trace back & trace forward
Text Requirements Editor
EUC Model
1. Validate requirements against objectives
2. Apply scenarios against requirements.
3. Perform ambiguity and domain expert review.
EUC Pattern Library
2
Trace back & trace forward
B
TR Pattern Library
Test Requirements
3
3
TC Pattern Library
Trace back & trace forward
4
Test Case
1. Generate test requirements and test cases
2. Consistency check
3. Review test cases
C
2
EUI Prototype Model
EUI Pattern Library
1. Validate/review the design and codes against test cases
2. Execute test cases against the code
Workable prototype
Test Scripts

Figure 3. The Overview of Our Proposed Approach

From the syntax rules, we have defined the phrase structure tree (PSTs) for the test requirements as described in Figure 4. Some examples of sentences that follow our test requirements' sentence structure are:

1. Validate (VB) that (Art) user (NN) can (MD) login (VB) with (Prep) valid (Adj) user name and password (NN).
2. Validate (VB) that (Art) user (NN) can (MD) withdraw (VB) the correct (Adj) amount (NN).

We then created the pattern library for the test cases, which consists of a few main components; such as test case ID, test requirements, test description, pre-condition, input/test data, steps, and expected result. Table 1 describes the detail description of each component for our test case pattern library. Our EUC model, test requirements and test cases are dependent to each other. Figure 5 and 6 describes the dependency in detail.

As depicted in Figure 5, one abstract requirement in the form of an EUC model can generate many test requirements. Each test requirements can generate one or many test cases, and one test script can execute one or many test cases. In this sample requirements (refer appendix), the EUC of "Identify self" can generate three test requirements and each test requirements can generate one or many test cases. Table 2 reflects the relationship of each component as depicted in Figure 5. This dependency is important to ensure each component can be trace-back and forth as depicted in Figure 7.

Figure 6 illustrates dependencies between the generated EUC and EUI model, test requirements and test cases that were considered while developing our pattern libraries. The item labeled with [1] is the EUC model, [2] is the EUI prototype model, [3] is the test requirements and [4] is the test case. As mention earlier, the EUC model is categorised into "Input" [A] and "Output" [B]. In this example the EUC of "Identify Self" is the "input" where it represents the action of the actor (user or system). This is reflected in the test requirements statement "user is able to login". Meanwhile the "Output" [B] from the EUC model is reflected in the "Expected Result" of the test case. Then we defined the input/test data of the test case based on stated condition, which is "valid username and password". The rule to define the test data is that it should be able to stress the expected application weaknesses. For example, we defined the test data for the password of the login module as *Admin00!*. This test data reflect to the common security password rules, where it combined the upper and lower cases, numbers and special characters.

Figure 6 also shows the dependency between the mock-up UI prototypes design with the EUI model. In this example, the EUI of "ID" is linked to the mock-up UI that labeled with (a). Meanwhile, the EUI model of "verify identity" and "offer choice" are linked to the same mock-up UI and labeled with (b) and (c) respectively. From here we defined the "Steps" or the test procedure for the test case.

We also provide a template editor to allow the user to modify the essential interaction of the EUC model, test requirements and test cases. However, this is not available for the mock-up UI prototypes pattern. Currently the mock-up UI prototypes are limited and linked to its relevant test requirements and test cases. Therefore, for each new test requirements and test cases inserted we have to design and develop the mock-up UI prototypes. To overcome this issue, we plan to integrate our tool with an integrated development environment (IDE) to allow for the source code and UI editing. This IDE is also expected to help the developer to review and update the generated test scripts.

Table 1. Our test case pattern library component

TC Component	Description
Test case ID	This is the test case identification number. Each number must be unique.
Test requirements	The related test requirements to the test cases. It is a statement that identifies what need to be tested and validated.
Test description	Defines the test cases/scenarios based on the test requirements. The statement can be with or without samples of input test data.
Pre-condition	Lists of conditions other than test case or system state that must be in place for this test case to run/execute.
Steps	The list of steps/flow of actions to execute the test case.
Expected result	Specifies all of the output and features from the indicated test requirements and also from the list of EUC and EUI prototype model.

Table 2. Sample of our abstract tests pattern

EUC Model	Test Requirements	Test Cases/Scenarios
Identify self	Validate that user is able to login with valid username and password.	Valid username and valid password.
	Validate that user is not able to login if username or password is invalid.	Valid username and invalid password.
		Invalid username and valid password.
		Invalid username and invalid password.

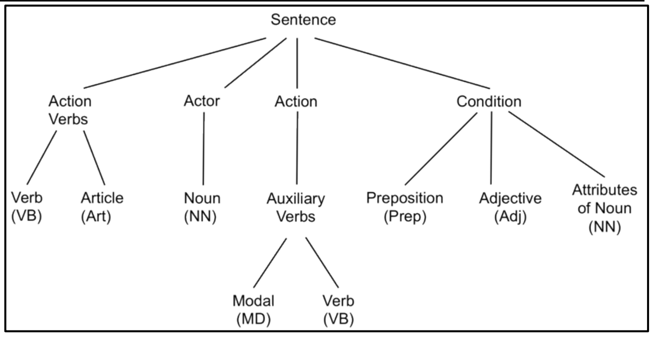


Figure 4. Phrase Structure Tree (PST) for our test requirements pattern library.

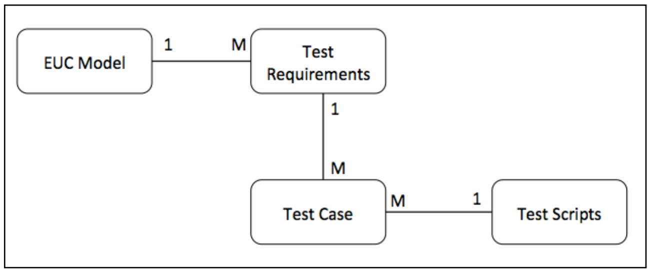
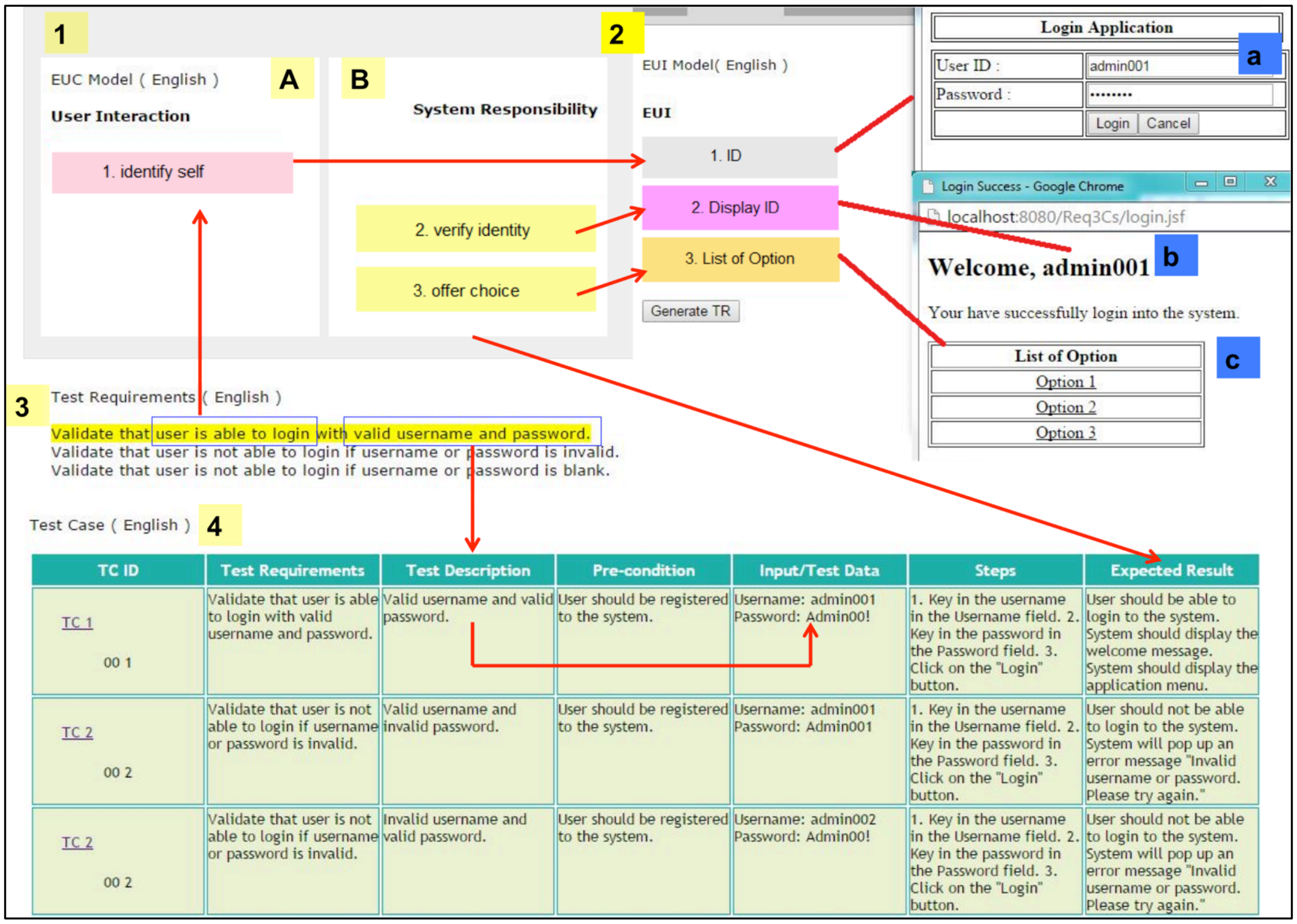


Figure 5. The dependency relationship between EUC model, test requirements, test cases and test scripts



Welcome, admin001

Your have successfully login into the system.

Validate that user is able to login with valid username and password.
Validate that user is not able to login if username or password is invalid.
Validate that user is not able to login if username or password is blank.

TC ID	Test Requirements	Test Description	Pre-condition	Input/Test Data	Steps	Expected Result
TC 1 00 1	Validate that user is able to login with valid username and password.	Valid username and valid password.	User should be registered to the system.	Username: admin001 Password: Admin00!	1. Key in the username in the Username field. 2. Key in the password in the Password field. 3. Click on the "Login" button.	User should be able to login to the system. System should display the welcome message. System should display the application menu.
TC 2 00 2	Validate that user is not able to login if username or password is invalid.	Valid username and invalid password.	User should be registered to the system.	Username: admin001 Password: Admin001	1. Key in the username in the Username field. 2. Key in the password in the Password field. 3. Click on the "Login" button.	User should not be able to login to the system. System will pop up an error message "Invalid username or password. Please try again."
TC 2 00 2	Validate that user is not able to login if username or password is invalid.	Invalid username and valid password.	User should be registered to the system.	Username: admin002 Password: Admin00!	1. Key in the username in the Username field. 2. Key in the password in the Password field. 3. Click on the "Login" button.	User should not be able to login to the system. System will pop up an error message "Invalid username or password. Please try again."

Figure 6: The dependency between EUC model, test requirements and test cases

4. USAGE EXAMPLE

Figure 7 illustrates the use of our prototype tool with a sample set of requirements for login behaviour (refer to Appendix A). Nancy, a requirements engineer would like to validate the requirements specification provided by the client-stakeholder. As shown in Figure 7, she inserts the requirements in the form of user scenario or use case narrative in the text editor (1). Then, she updates the model to generate the EUC model (2) from the textual requirements (1). From there, she clicks the "Create EUI" button to generate the low-fidelity EUI model (3). Here, she can perform the initial ambiguity review to validate the correctness of the generated EUC and EUI models. She also can perform the traceability check by clicking on the EUC and EUI model component. She clicks on the EUI of "ID" to trace-back to its relevant EUC model. Then, she clicks on the EUC of "Identify self" to trace-back its essential interaction. In this example the essential interaction for the EUC model of "Identify self" is "login".

Next, she clicks the "Generate TR" button to generate the associated test requirements (4). She then clicks the "Generate Test Case" button to generate the related test cases (5). Here, she can review correctness of the generated test requirements and test cases. Then, she clicks one of the test requirements to populate the associated mock-up UI prototype (6). She tests the UI prototype by providing the input data given in the test case and see the expected results. She can then indicate the result of the testing in the "Testing Status" column (7). Test results are saved in a database and can be retrieved for future reference. This process can be performed in collaboration with all client-stakeholders. Any incomplete or inconsistent requirements or test cases generated in this process can be expressed during the process.

5. EVALUATION

We have conducted evaluations to investigate the utility of our TestMEReq tool in validating requirements. First we evaluated the accuracy of the tool in extracting accurate abstract tests from 15 requirements sample of various software application systems. Then we conducted a usability study and interview with three requirements engineers as well as 79 undergraduate students.

5.1 Accuracy of the Abstract Tests Pattern

We evaluated the ability of TestMEReq in producing accurate abstract tests. We calculated the ratio and the average in order to see the performance of the tool to extract the abstract test. We evaluated its accuracy when applied to nearly 100 requirements scenarios from requirements sample of 15 software application such healthcare, banking and rental system.

Figure 8 shows the correctness ratio for TestMEReq for each software application system. This shows some variability across the range of scenarios, but the average correctness across all scenarios and interactions is approximately 84%. Our automated tool does not (and cannot) produce 100% correct answers due to the incomplete generated EUC and EUI models as they depend on the correct extraction from textual requirements which, as demonstrated in our earlier work, may face some issues due to language and sentence structures. This implies that users must have knowledge of how to write good user scenarios or use case narratives in order to generate a complete EUC and EUI model.

5.2 Expert Review

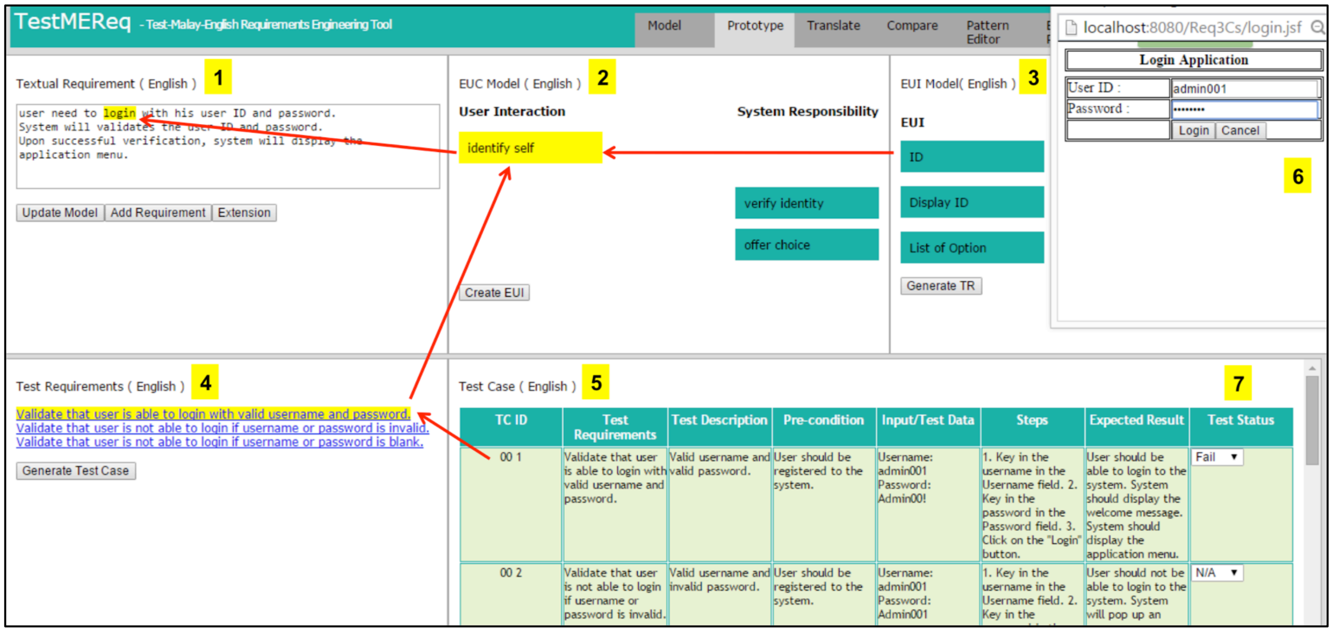
We conducted interviews with three expert requirements engineers in order to get their opinions regarding the usability of our prototype tool. All of them have more than five years working experience in the information technology (IT) industry. During the interview, we briefed the experts (identified below as E1, E2, and E3) on the purpose of the interview, and defined the different terminologies and definitions used in our interview questions to enhance consistency of responses. We provided a brief description and explained the main objectives of our prototype tool and gave them access to a link for them to explore the tool with sample requirements.

From the interview, all of the reviewers were agreed that our prototype tool is helpful in validating and clarifying users requirements through the generated abstract tests and mock-up UI prototype. The automatic approach helps to reduce time and effort to generate the abstract test. The generated abstract tests also help to trigger ideas to the client-stakeholders on what they want to test and achieve with the requirements. They also agreed that the tool is simple, easy to use and learn. However, they pointed out some limitations and gave suggestions for us to improve the tool. The first expert (E1) suggested that an animation preview of our tool should be included. This may help the user with an initial overview of the running prototype based on the generated test cases. Expert E2, commented that "It is a good approach as the tester can participate at the earlier phase of SDLC. In current practice, testers are not involved until at the later stage of the development process." For further improvement, he suggested that we define the rules or language pattern for the textual requirements entered by the user. This may help to avoid ambiguity in the initial requirements. Expert E3, recommended that we add another feature to our tool: a template of test requirements and test cases. He noted that this feature may help users to modify and add new test requirements and test cases that are not in our pattern library. This also may help to further improve and expand our test requirements and test case pattern library.

5.3 Usability Study

We conducted a usability evaluation with 79 final year undergraduate students, majoring in Software Engineering and enrolled in a Software Testing course. This study is to evaluate the usability of our tool to generate abstract test for validating requirements. The participants were requested to perform two tasks: the first task is to explore the tool with a provided sample requirement (refer Appendix A), and the second task is to complete a survey questionnaire. The participants were informed that they would be observed and encouraged to speak aloud their views of the tool while completing the task. The purpose of the observation was to identify problems and misconceptions faced by the participants when using the tool. The verbal evaluation of the tool provided us with the users' spontaneous responses and suggestions for improvement as they use the tool. After the completion of the task, students were requested to answer five questions related to the usability of the tool. They responded to five questions regarding the usefulness, ease of use, ease of learning, and satisfaction of the tool based on a five-level Likert scale.

Figure 9 shows the results of the participant survey of the tool usefulness, ease of use, ease of learning and satisfaction. In terms of the usefulness of the tool, 86% of the participants found that the tool is useful, 89% agreed the tool is easy to use, 87% agreed that the tool is easy to learn and 80% were satisfied with the tool. Overall, the usability results indicate that our prototype tool is useful, easy to use, easy to understand, and able to satisfy users. Although they also agreed that the tool is simple and easy to use, they would like to have a better user interface and a simple user guide to use the tool. A small number of the students encountered some difficulty with the tool's test case pattern library due to the inability of the tool to handle more complex requirements.



```
user need to login with his user ID and password.
System will validates the user ID and password.
Upon successful verification, system will display the
application menu.
```

Validate that user is able to login with valid username and password.
Validate that user is not able to login if username or password is invalid.
Validate that user is not able to login if username or password is blank.

TC ID	Test Requirements	Test Description	Pre-condition	Input/Test Data	Steps	Expected Result	Test Status
00 1	Validate that user is able to login with valid username and password.	Valid username and valid password.	User should be registered to the system.	Username: admin001 Password: Admin00!	1. Key in the username in the Username field. 2. Key in the password in the Password field. 3. Click on the "Login" button.	User should be able to login to the system. System should display the welcome message. System should display the application menu.	Fail
00 2	Validate that user is not able to login if username or password is invalid.	Valid username and invalid password.	User should be registered to the system.	Username: admin001 Password: Admin001	1. Key in the username in the Username field. 2. Key in the	User should not be able to login to the system. System will pop up an	N/A

Figure 7. Usage example of TestMEReq. The red arrow shows the traceability check between test cases, test requirements, EUC and EUI model and textual requirements

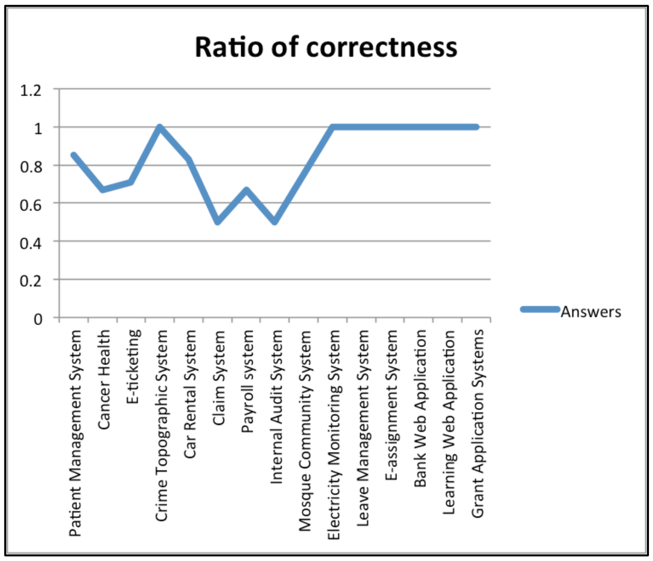
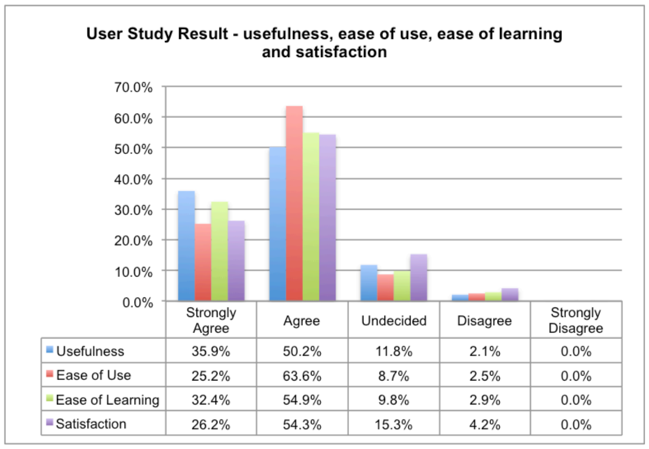


Figure 8. Results of tool correctness



	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
🟦 Usefulness	35.9%	50.2%	11.8%	2.1%	0.0%
🟥 Ease of Use	25.2%	63.6%	8.7%	2.5%	0.0%
🟩 Ease of Learning	32.4%	54.9%	9.8%	2.9%	0.0%
🟪 Satisfaction	26.2%	54.3%	15.3%	4.2%	0.0%

Figure 9. Usability study result

6. CONCLUSION AND FUTURE WORK

TestMEReq is a tool that assists the requirements validation process between RE and client-stakeholder. Our tool integrates the semi-formalised abstract EUC and EUI models with requirements-based testing and rapid prototyping techniques. Our tool automatically generates the abstract tests and the mock-up UI prototypes from EUC and EUI models. The generated abstract tests describe the tested functionality of the requirements. Meanwhile, the UI prototype provides a visualisation of the requirements based on the generated test cases. These two main components help the RE and other stakeholders to gain a better understanding of their requirements. Our studies suggest that the automated support provided by our tool helps to reduce the time and effort in validating requirements through the generation of the abstract test cases and visualisation of the UI.

For future work, we plan to embed a requirements prioritisation method to our tool for prioritising generated tests. This will help the user to organize the validation of the requirements based on the generated test cases. Furthermore, we intend to enhance this tool as a collaborative validation tool to allow better communication and discussion between the client-stakeholders and REs across different geographical locations.

7. ACKNOWLEDGMENTS

This research is funded by Ministry of Higher Education Malaysia (MOHE), Universiti Teknologi Mara (UiTM), Fulgent Corporation and FRGS grant: FRGS/2/2013/ICT01/FTMK/02/2/F00185.

8. REFERENCES

[1] M. Kamalrudin, “Automated software tool support for checking the inconsistency of requirements,” *ASE2009 - 24th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 693–697, 2009.

[2] C. Denger, D. M. Berry, and E. Kamsties, “Higher quality requirements specifications through natural language patterns,” in *International Conference on Software Science, Technology and Engineering (SwSTE)*, 2003, pp. 1–11.

[3] F. Fabbrini, M.Fusani, S.Gnesi, and G.Lami, “The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool,” *Softw. Eng. Work. 2001. Proceedings. 26th Annu. NASA Goddard* , pp. 95–105, 2001.

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

[5] U. A. Raja, “Empirical studies of requirements validation techniques,” in *2009 2nd International Conference on Computer, Control and Communication, IC4 2009*, 2009, pp. 1–9.

[6] R. Biddle, J. Noble, and E. Tempero, “From Essential Use Cases to Objects,” in *forUSE 2002 Proceedings*, 2002, vol. 1, no. 978.

[7] L. L. Constantine and L. A. D. Lockwood, “Structure and Style in Use Cases for User Interface Design,” vol. 1, no. 978. Addison-Wesley Longman Publishing Co., Boston, MA, 2001.

[8] S. W. Ambler, “Essential (Low Fidelity) User Interface Prototypes,” 2003. [Online]. Available: http://www.agilemodeling.com/artifacts/essentialUI.htm.

[9] M. Kamalrudin, J. Hosking, and J. Grundy, “Improving requirements quality using essential use case interaction patterns,” *2011 33rd Int. Conf. Softw. Eng.*, pp. 531–540, 2011.

[10] M. Kamalrudin, J. Grundy, and J. Hosking, “Tool support for essential use cases to better capture software requirements,” *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng. - ASE ’10*, p. 255, 2010.

[11] M. Kamalrudin, N. A. Moketar, J. Grundy, and J. Hosking, “Automatic Acceptance Test Case Generation From Essential Use Cases,” in *13th International Conference on Intelligent Software Methodologies, Tools and Techniques*, 2014, pp. 246–255.

Appendix A: Sample of Requirements (System Use Case)

1. User needs to login with his user ID and password.
2. System will validate the user ID and password.
3. Upon successful validation, the system will display the application menu.