3rd Asia-Pacific Requirements Engineering Symposium (APRES) 2016, 10-12 November 2016, Nagoya, Japan, (c) Springer 2016

# Automated Support to Capture and Validate Security Requirements for Mobile Apps

Noorrezam Yusop[1], Massila Kamalrudin[1], Safiah Sidek[1], and John Grundy[2]

[1]Innovative Software System and Services Group,
Universiti Teknikal Malaysia Melaka, Malaysia
p031320001@student.utem.edu.my,{massila,
safiahsidek}@utem.edu.my

[2]School of Information Technology, Deakin University,
Geelong, Australia
j.grundy@deakin.edu.au

**Abstract.** Mobile application usage has become widespread and significant as it allows interactions between people and services anywhere and anytime. However, issues related to security have become a major concern among mobile users as insecure applications may lead to security vulnerabilities that make them easily compromised by hackers. Thus, it is important for mobile application developers to validate security requirements of mobile apps at the earliest stage to prevent potential security problems. In this paper, we describe our automated approach and tool, called MobiMEReq that helps to capture and validate the security attributes requirements of mobile apps. We employed the concept of Test Driven Development (TDD) with a model-based testing strategy using Essential Use Cases (EUCs) and Essential User Interface (EUI) models. We also conducted an evaluation to compare the performance and correctness of our tool in various application domains. The results of the study showed that our tool is able to help requirements engineers to easily capture and validate security-related requirements of mobile applications.

**Keywords:** security requirements; security attributes; validation; test driven development; mobile apps; model based testing strategy; euc; eui.

## 1 Introduction

Mobile phones have been used widely as they allow interactions between people and things anywhere and anytime. The use of mobile application is rapidly growing, especially in performing online transactions, such as online purchasing, flight booking and hotel booking. There are also a plethora of applications being developed to fulfil the needs of mobile users. However, many mobile application developers tend to ignore the security aspect of the application during the early stage of development, leading to malicious attacks and security breaches. It is also found that most of the requirements engineers fail to capture correct security related requirements during the elicitation

phase as they face difficulties to understand the terms and knowledge of the security [1]. Further, the process of capturing correct and consistent requirements from client-stakeholders is often difficult, time consuming and error prone [2][3]. Therefore, there is a need for automation support to capture and validate security related requirements at the early stage of mobile application requirements engineering. In our previous work [4], we have conducted a user study to gauge requirements engineers' ability in capturing the security related requirements from a set of business requirements of a mobile application. The study found that the participants captured almost 60% incorrect security attributes for each of the requirements given. This result indicates that requirements engineers face difficulty to capture the security related requirements, especially in extracting the security attributes [4][5]. Further, it was found that the longest time taken by the participants to extract the security attributes is more than 45 minutes, which means that more effort is needed to perform this task. These challenges have motivated us to: 1) develop an automated tool support for capturing and validating security requirements, and 2) evaluate the tool to demonstrate its ability to enhance the accuracy and usability for capturing and validating security requirements of mobile apps.
This paper describes the approach and an automated that captures and validates security requirements for mobile applications using Test Driven Development (TDD) with a model-based testing strategy using the Essential Use Cases (EUCs) and the Essential User Interface (EUI) models. We present background for this study, our prototype tool, and an experiment comparing its performance in extracting security attributes and validation from security requirements. Finally, we discuss implications and future work.

## 2 Background

### 2.1 Test Driven Development (TDD)

Test-driven development (TDD) is a development strategy that has been popularized by extreme programming [6]. The three important stages in TDD are (1) writing the test before adding the code, (2) writing the simplest code that passes the test, and (3) repeating this cycle until the software is matured. TDD also allows each of the requirements to be transformed to a test and it helps the engineers to think through the requirements or design before writing the functional code. TDD promotes a style of incremental development, where it identifies any behaviour that has been correctly implemented or remains undone. This approach enhances the analysis and the design of software as it allows the software to be tested at any time under automation [7].

### 2.2 Essential Use Cases (EUCs)

The EUC approach was defined by Constantine and Lockwood as a "structured narrative, expressed in a language of the application domain and of users, comprising a simplified, generalized, abstract, technology free and independent description of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in a role or some roles in relation to a system and that embodies the purpose or

intentions underlying the interaction" [8]. Its main objectives are to support better communication between the developers and stakeholders via a technology-free model and to assist better requirements capture. These objectives can be achieved by allowing only specific details relevant to the intended design to be captured [9]. 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 realizations that allow users to rethink, or reengineer the aspects of the overall business process. Figure 1 shows an example of natural language requirements (left) and an example of EUC (right) when capturing the requirements. The natural language requirements (highlighted) are shown on the left hand side.

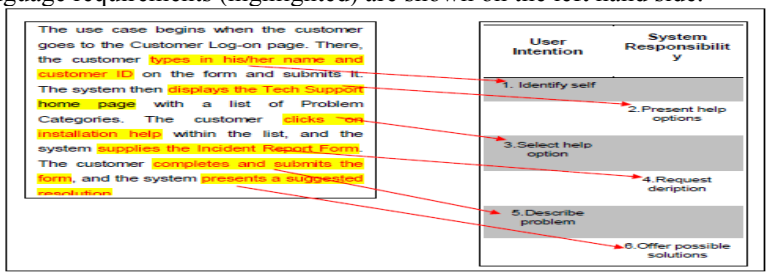


**Fig. 1.** l Natural Language Requirements (left) and Essential Use Case (EUC) (right) [8][10]

When capturing requirements from natural language text, the EUC model is found to be more suitable than the conventional UML use case. An equivalent EUC description is generally shorter and simpler than a conventional UML use case as it only comprises the essential steps (core requirements) of user's intrinsic interest. It contains the user's intentions and the system responsibilities to document the specific interaction without the need to describe the user's interface in detail. It is reported in [5] that EUCs are beneficial for capturing security requirements.

### 2.3 Essential User Interface (EUI)

EUI prototyping is a low fidelity prototyping approach [11]. It provides the general idea behind the UI instead of its exact details. Focusing on the requirements rather than the design, it represents UI requirements without the need for prototyping tools or widgets to draw the UI [12]. EUI prototyping extends from and works in tandem with the semi-formal representation of EUCs that also focuses on the users and their usage of the system, rather than the system features [13]. It thus helps to avoid clients and REs from being misled or confused by chaotic, evolving and distracting details. EUI also allows some explorations of the usability aspects of a system. Figure 2 shows examples of EUI prototype developed from EUC models.

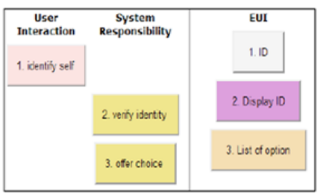


**Fig. 2.**Examples of EUI prototype from EUC models

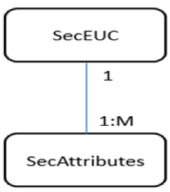


**Fig. 3.** The relationship between SecEUC model and SecAttrributes

**Table 1.** Example of SecEUC Pattern Libraries

| SecEI | SecEUC | SecEUI | SecCtrl |
|---|---|---|---|
| Check username | Identify self | ID | Authentication |
| Check password | | | |
| Verify username | | | |
| Make payment | Make payment | Payment Type | Transaction |
| Complete payment form | | | |

### 2.4 SecEUC and SecEUI

SecEUC is a security pattern library comprising security related EUC, while SecEUI is the security related EUI. Yahya et al. [5] have developed the security pattern library, called the SecEUCs and security related essential interaction termed as the SecEI pattern library. Examples of the SecEI and the SecEUC are shown in Table 1. They used EUC model to capture security requirements to allow requirements engineers to identify and capture the security requirements. This pattern library recognises that there is a direct relationship between the SecAttributes and SecEUC pattern library. As shown in Figure 3, the relationship is one SecEUC to many SecAttributes (one to many). The main purpose for choosing the SecEUC and SecAttributes pattern library as well as their model is to conduct an in-depth analysis that could help to capture and validate security requirements from the business requirements. The SecEUC library patterns are based on EUCs generated from normal business requirements, while SecEI library patterns are based on the essential interactions found in security-related requirements.

Currently, both patterns only support the software/system development, but not the mobile application development. Further, the development of SecEUC patterns was adapted from the works of [14][15][16] and the identification of associated security elements are based on the definitions from the basic security services.Table 1 shows the example of SecEUC pattern libraries together with the relationship between many SecEI to one SecEUC and SecEUI. As shown in Table 1, the three SecEI identified as “check username”, “check password” “verify username” are related to one SecEUC “Identify self”, which is related to one SecEUI “ID”. Similarly, the two SecEI, which are “Make payment” and “Complete payment form” are related to one SecEUC “Make payment”, which is related to one SecEUI “Payment Type”.

## 3 Our Approach

The purpose of this study was to develop an approach and an automated tool to assist requirements engineers to automatically capture and validate the security-related requirements of mobile application. Our key research questions were: 1. Can an automated approach using TDD methodology with EUC and EUI models able to facilitate the extraction and validation of security attributes for security related requirements of mobile application? 2. Does the automation incorporated in the tool allow requirements engineers to quickly and accurately capture and validate the security attributes for security related requirements of mobile application? 3. How do the target users evaluate the usefulness of the automation tool in facilitating the extraction of security attributes and validation of the security requirements of mobile application?

Guided by these research questions, we developed an approach and a tool support, MobiMEReq, that automatically capture and validate the security requirements of mobile application. Here, the mobile application specific concerns and characteristics that reflected to this proposed approach are authentication, authorization and confidentiality. As shown in Figure 4, our approach is depicted in the box labeled as A and our tool support is depicted in the box labeled as B. In this approach, we adopted the TDD methodology with a model-based testing strategy using EUCs and EUI models.

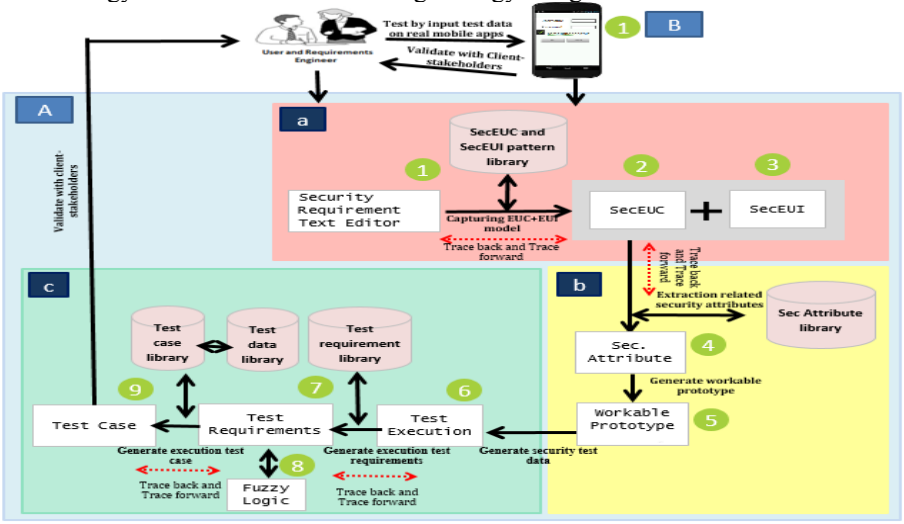


**Fig. 4.** Overview of our approach

As shown in Figure 4, the approach is divided into three stages: capture the security requirements (a), generate the security attribute (b), and generate test (c). The first stage of our approach begins when the textual requirements are analyzed and traced to the EUCs patterns library for appropriate abstract interaction in a form of EUC model (1). Then, the SecEUC (2) and SecEUI (3) are derived from the generated EUC models based on the categorization of their attribute related to the security element as defined in the SecEUC pattern library. The second stage involves generating the security attributes. At this stage, each security attribute is generated from the SecEUC and SecEUI based on a defined security attribute library (4). Next, a workable prototype is generated to visualize the security requirements based on the generated SecEUI (5). This helps to validate the captured requirements with the textual captured requirements. Further, the security requirements are validated by the generated test that comprises test requirements (7) and test cases. This validation approach can also be done reversely from the generated and workable application prototype run in mobile to MobiMEReq (1) where all the test components, security requirements attributes as well as EUC and EUI models are traced back. To realize our approach, we developed two pattern libraries, namely the mobile SecAttributes Pattern library and the Mobile Security Pattern Library. This approach also adopted the concept of fuzzy logic to prioritize the test requirements and test cases to validate the security requirements. The following section describes the development of these two libraries and the test prioritization.

### 3.1 Mobile SecAttributes Pattern Library

We developed the Mobile SecAttributes Pattern Library [17], consisting of SecEUC, SecEUI and related security attributes for mobile apps. Here, a security attribute is defined as any piece of information that may be associated with a controlled implicit entity or user for the purpose of implementing a security policy. Then, the SecEUC and SecEUI in this library were derived from a collection of security attribute requirements of mobile apps from industry security requirements and other published material [18].

### 3.2 Mobile Security Pattern Library

We developed a mobile security pattern library to support the extraction of the security related attributes from the security requirements. This pattern library consists of SecAttributes patterns with Test requirements and test cases. As shown in Figure 5, it is found that one SecEUC is associated with one to many security attributes and many test requirements. In this case, one test requirements is associated with many test cases. For now, we have stored almost 280 of sec attribute and 185 of test requirements and 370 test cases. They are all from the requirements collected from industry and real projects. Examples of our mobile security pattern library are shown in Table 3.

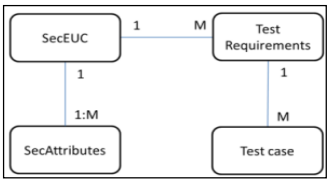


**Fig. 5.** The relationship of SecEUC with Sec Attributes and Test Requirements

**Table 2.** Sample of our SecAttributes Pattern and Test Requirements

| SecEUC | SecAttributes | SecCtrl | Test Requirements | Test case | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Test description | Pre-Condition | Test data | Steps | Expected Result | Actual result | Test Status |
| Make payment | Username | Authentication Transaction | validate that user can make payment | Validate that user can make payment if user account exist in database. | User require to login | Username='noorrezam' Password='noorrezam123' PaymentID='P11223344' | 1.Login 2.Key in Amount and click Submit button | User will be able to make payment to the system. | User successful to make payment to system. | PASS |
| | Password | | Validate that user cannot make payment | Validate that user cannot make payment if user account exist in database. | User require to login | Username='noorrezam' Password='noorrezam123' Amount='0' PaymentID='P11223344' | 1.Login 2.Key in Amount and click Submit button | User will be able to make payment to the system. | User unsuccessful to make payment to system. | FAIL |
| | PaymentID | | | | | | | | | |
| Verify TAC Code | Username | Authentication Authorization Transaction | Validate that user can verify TAC code | Validate that user can insert TAC Code if user account exist in database. | User require to login | Username='noorrezam' Password='noorrezam123' TAC Code ID='TAC112' PaymentID='P11223344' | 1.Login 2.Key in TAC CODE ID and click Submit button | User will be able to verify TAC Code to the system. | User successful to verify TAC Code to system. | PASS |
| | Password | | | | | | | | | |
| | TAC Code ID | | Validate that user cannot verify TAC code | Validate that user cannot insert TAC Code if user account exist in database. | User require to login | Username='noorrezam' Password='noorrezam123' TAC Code ID='TAC112' PaymentID='P11223344' | 1.Login 2.Key in TAC CODE ID and click Submit button | User will be able to verify TAC Code to the system. | User successful to verify TAC Code to system. | FAIL |
| | Payment ID | | | | | | | | | |

### 3.3 Test Prioritization using Fuzzy Logic

Considering that a test case prioritization approach is able to improve the rate of faulty detection during the testing phase [19], we adopted the concept of test prioritization to prioritise the test requirements and test cases to validate the security requirements. In this case, the level of importance of the test case that runs during the test execution is identified according to the scale of high, medium and low, which has been validated by the experts. Further, the test case prioritization should be based on the captured requirements. In cases where the requirements are not prioritized, the test group is required to propose the prioritization to clients for reviews. To do this, Fuzzy logic is adopted to our work. Fuzzy logic uses the 'uncertainty principle' that recognizes the use of an approximation rather than a fixed or exact value, which uses a range of true values instead of true or false value. It also uses a form of many-valued logic that has different ranges of membership between 0 and 1 defined by a fuzzy set [20]. This approach helps to automatically select the best test cases for each of the test requirements and helps to reduce the number of test cases [21].Figure 6 shows the step-by-step procedure of applying the fuzzy logic to conduct test prioritization in our security requirements validation work. The algorithm as shown in Pseudo-code1 was applied to prioritize the test case from the generated test requirements.

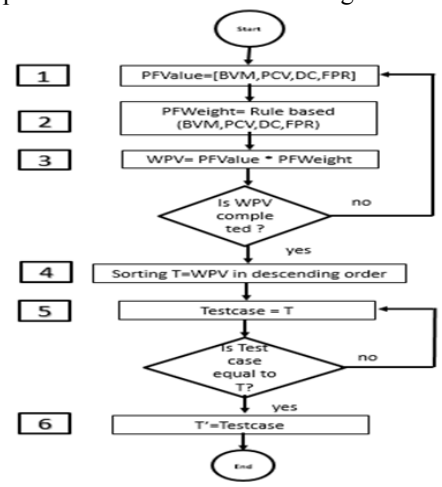


**Fig. 6.** Test requirement for prioritization test case algorithm Flow chart

**Pseudocode 1: Pseudo code algorithm to prioritise test case**

Pseudocode algorithm.

The set of Test Requirement is denoted by T
For each Test Requirement
Calculate Weighted Prioritization Value (WPV) using eqn (1)
End For
Arrange by sorting T in descending order
For each test case
Choose test case based on T (Test Requirement)
End For
Output:
Our final reduced test cases is T'

The step by step to prioritize test case is shown in Figure 6 and described below:
Step 1: To assign weight to the requirements, four prioritization factors (PF) are considered using fault severity as proposed by Kumar et al. [19]. We use this factor as an input parameter to the security requirements to embed in our fuzzy logic for target test requirements. These factors are: i. Business Value Measure (BVM): BVM is a measure, in which security requirements with the highest level of importance are the critical requirements to customer's business. The range of each requirement is from 1 (low) to 10 (high). Use cases can also be used to analyze the security requirements. ii. Project Change Volatility (PCV): PCV is based on how many times a customer modifies the project security requirements during the software development cycle. PCV is one of the

criteria that help to assess the changes in the security requirement at the early stage after implementation of project is start. PCV is increase test efforts and the project is difficult to complete on time. iii. Development Complexity (DC): Each security requirement is analyzed based on the complexity of its implementation. Factors, such as development efforts, technology, environmental constraints and security requirement feasibility matrix are considered when measuring the complexity of the implementation. iv. Fault Proneness of Requirement (FPR): FPR is a measurement based on the error prone includes number or occurrence of in-house test failures found and also security requirements failures reported by the customer. Table 4 below show example an assignment low-high value for related test requirements based on four input factor.

| Range | Low(1)-High (10) | | | |
|---|---|---|---|---|
| **Test Requirements** | **BVM** | **PCV** | **DC** | **FPR** |
| validate that user can register to the system | 2 | 10 | 3 | 6 |
| validate that user can login to the system | 2 | 5 | 5 | 5 |

**Table 3.** Low-High Assignment Test Requirements

Step 2: Figure 6 [2] shows the Priority Factor (PF) ruled based used as four input parameters to assign Priority Factor Weight. As shown in Table 5, there are: 4 parameters = BCM, PCV, DC, FPR. 3 memberships = Low, Medium, High; Rule-Based= 4 the power of 3= 4x4x4=64 rules. Thus, based on the range applied in this study, 64 rules or less can be used. The pseudo-code algorithm for the rule based in our fuzzy algorithm is shown in Pseudo-code 2 below.

**Table 4.** Weight Prioritization value based on Test Requirements

| | BCM | PCV | DC | FPR | WEIGHT |
|---|---|---|---|---|---|
| 1 | LOW | LOW | LOW | LOW | LOW |
| 2 | LOW | LOW | MEDIUM | LOW | LOW |
| 3 | LOW | LOW | HIGH | LOW | LOW |
| : | : | : | : | : | : |
| 18 | MEDIUM | HIGH | HIGH | HIGH | HIGH |
| : | : | : | : | : | : |
| 63 | HIGH | MEDIUM | HIGH | HIGH | HIGH |
| 64 | HIGH | HIGH | HIGH | HIGH | HIGH |

**Pseudocode 2**: Rule-based Pseudo code algorithm

```
If (BVM is low && PCV is low && DC is high
&& FPR is low) Then Weight is low
If (BVM is low && PCV is medium && DC is
high && FPR is low) Then Weight is low
If (BVM is low && PCV is low && DC is low
&& FPR is low) Then Weight is low
If (BVM is high && PCV is high && DC is high
&& FPR is high) Then Weight is high
End IF
```

Step 3: Based on the four input parameters provided in the security requirements and the rule based, fuzzy inference system and defuzzification are then used to prioritize the test cases. The metric equation used for the prioritization of the test case is shown below.

WPV= $(\sum_{pf=1}^{h} PFvalue * PFweight)$…….. (1)

where, WPV is the weightage prioritization for each test case calculated based on the four input parameters. PF value is the value assigned to each test case. PF weight is the weight assigned for each input parameter. The weights are obtained from the fuzzy

rules and the WPV is calculated from Eq(1) which in turn gives the value of the prioritization order.

WP [TestReq1] = 2x 0.25+5x 0.25 + 5x 0.25+5x0.25=4, WP [TestReq2] = 2x 0.25+10x 0.25+3x 0.25+6x0.25=5.5,WP [TestReq3] = 2x 0.25+10x 0.25+3x 0.25+3x0.25=4.5

Step 4: Table 6 shows the sample results of the weight prioritization of test cases for the three functional requirements from the test requirements. Based on the calculation, the descending values of WPV are TestReq2, TestReq3, and Test Req1.

**Table 5.** Samples of Weight Prioritization value based on Test Requirements

| Factor / Test Requirements | TestReq1 | TestReq2 | TestReq3 | Weight |
|---|---|---|---|---|
| BVM | 2 | 2 | 2 | 0.25 |
| PCV | 5 | 10 | 10 | 0.25 |
| DC | 5 | 3 | 3 | 0.25 |
| FPR | 5 | 6 | 3 | 0.25 |
| Weight Prioritization Value | 4 | 5 | 4.5 | 1 |

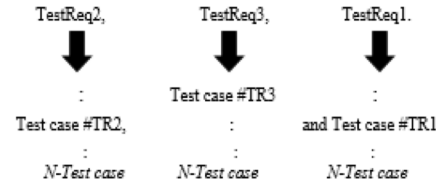


**Fig. 7.** Selection test case from test requirements

Step 5: The selection number of test case from test requirements is selected as shown Figure 8 based on the test requirements from ordering of WPV (Test Requirement).

## 4 Implementation

We have developed a prototype tool called MobiMEReq to realise the approach of automatically capture and validation of security requirements as discussed in the previous section. This prototype tool is an extension of our earlier [14] tool that runs in both mobile and web applications. Figure 8 shows the tool usage based on the overview of our approach as shown in Figure 4 and the role of the SecAttributes Pattern Library and the embedded fuzzy logic in validating the security requirements of mobile apps at the early stage of requirements validation. By implementing the TDD methodology and model based testing strategy, our approach is divided into two main parts: 1) Capturing security attributes from a set of mobile application requirements and 2) Validating the quality of security requirements.

1. Capturing the Security Attributes [A]: Here, the Mobile SecAttributes Pattern Library is used by the tracing engine to analyse the textual mobile requirements and then match it to the set of abstract interactions of SecEUC (A1). This approach allows Requirements Engineers (REs) to capture the important security attributes from the textual mobile requirements gathered from client-stakeholders A(1). Then, the textual requirements are mapped to SecEUC A(2) and SecEUI model A(3). As shown in second stage, Figure 4 A[b], the SecAttributes is generated to visualize the security attributes (Figure 8 A(4)) that best fit to the generated SecEUC and SecEUI model as described at first stage, Figure 4 A[a] based on the defined attributes in the Mobile SecAttributes Pattern Library. Next, the RE can visualise the security requirements as a form of workable rapid prototype model of the targeted mobile app [14].

2. Validating the quality of the security requirement consist of two parts: Part 1. Validation from MobiMEreq workable prototype [A].RE can visualise the security requirements in a form of workable rapid prototype model of the targeted mobile app. This allows the client-stakeholder to check the quality of the generated security requirements in terms of its correctness and consistency with the original security related requirements. Further, to validate the correctness of the security requirements, RE can insert the test data into the workable prototype, Figure 8 A(5). As shown Figure 4, the results of the test review (labelled as A[c]) allow the execution of the test data and the display of the test status. RE can then perform the mobile Security Execution (SecExec) A[c](6) (Figure 8, A(6)). Here, Test requirements A[c](7) are used based on the associated SecEUC. In order for RE to view the results of the test validation, the test cases A[c](9) are generated to visualise the validation based on the proposed fuzzy logic approach A[c](8). The results of the validation are also displayed as Pass or Fail as shown in Figure 8, labelled as A(8) for each of the generated test data. Part 2. Validating the generated mobile apps with MobiMEReq [B]: Another utility provided by our MobiMEReq tool allows the generated mobile apps to be validated reversely to MobiMEReq. Both the RE and client-stakeholder can validate the functionality of the apps by inserting a random test data through their mobile apps and the associated components such as test cases A(7), test execution A(6), security attributes A(4) and both EUCs A(3) and EUI model A(2) and textual requirements A(1) as shown in Figure 8. For large scale can view in this link[1].


**Fig. 8.** Example of tool usage for integration security attributes and visualization tool

[1] https://drive.google.com/drive/folders/0B5QVa-tMkodvNXZnVlc3SGxMbkE

# 5 Evaluation

Designed to be used by requirements engineers, we have conducted three studies to evaluate the accuracy and usability of our new automated tool in capturing and validating security requirements of mobile application. First, we conducted an accuracy test to evaluate the accuracy of the tool with three participants to check manually by applying a new set of security requirements. Secondly, we conducted a usability study with 50 undergraduate students to get their feedback based on four aspects, namely the usefulness, ease of use, ease of learning and satisfaction. Here a flight booking requirements that comprises security requirements such as login, payment and TAC code is used for the study. Finally, we requested two experts in the field of requirements engineering to test our tool and were later interviewed to collect their feedback.

## 5.1 Accuracy Test

We evaluated the accuracy of the tool by applying a new set of security requirements to check the false positive rate for the tool. False positive rate is used to calculate the ratio of the pass and fail for particular test. All of the related Test requirements were generated and they produced all correct false positive rate based on the prioritization of both test requirement and test case. The accuracy is measured based on the false-positive result of security requirements that provided by the tool compared to the sample answer prepared by the authors. The sample answer was first verified by the expert in the field of software engineering. Based on the results of the accuracy test shown in Table 7 (refer link to the accuracy test [2]), the MobiMEReq tool reported 100% correctness ratio based on the ten test requirements applied in the tool. This result indicates that the tool is able to facilitate requirements engineers to validate security requirements at the early stage of Software Development Life Cycle (SDLC).

**Table 6.** Sample of correctness of the automated validating tool

| Test Requirements | Pre-Condition | Test data | Test Status (Pass/Fail) | Expected Result | Actual Result | No.Correct answers | No.Wrong answers |
|---|---|---|---|---|---|---|---|
| Validate that user can login | Register | Username='noorrezam' Password='noorrezam123' | P | User can login and welcome page is displayed | User can login and welcome page is displayed | 1 | 0 |
| Validate that user can login | Register | Username='noorrezam' Password='noorrezam123456' | F | User can login and welcome page is displayed | User cannot login and warning message is displayed | 1 | 0 |
| Validate that user can inserting TAC Code | Login | Username=noorrezam Password=noorrezam123 TAC Code='123456' | P | User can inserting TAC Code and warning message is displayed | User can insert TAC Code and warning message is displayed | 1 | 0 |
| Validate that user can inserting TAC Code | Login | Username=noorrezam Password=noorrezam123 TAC Code='123234' | F | User cannot inserting TAC Code and warning message is displayed | User cannot insert TAC Code and warning message is displayed | 1 | 0 |
| Correctness ratio | | | | | | 10 | 0 |
| | | | | | | 100% | 0% |

## 5.2 Usability study

A survey to investigate the usability of the tool was also conducted with 50 participants. The questionnaire[3] was designed to gather participants' feedback regarding its usefulness, ease of use, ease of learning and satisfaction. In addition, the participants

[2] https://drive.google.com/drive/folders/0B5QVa-tMkodvb2ZuX3ROMzRGUWc

[3] https://drive.google.com/drive/folders/0B5QVa-tMkodvYlVDQk9uelc0X1k

were requested to write their comments on the four aspects. Results of the study are shown in Figure 9. With respect to usefulness, 80% of the participants felt that the tool is useful for capturing and validating security requirements. Only 1% of the respondents disagreed that the tool is useful and 19% of the participants were neutral. These results indicate that the majority of the participants agreed that the tool is useful. They also recommended that the tool need to be improved for better visualization so that users can view the summary result of test case at early of process of SDLC. With respect to ease of use, 77% of the participants agreed that the tool is easy to use, while only 5% disagreed with the statement. 18% of the respondents were indifference. These results also indicate that the majority of the participants agreed that the tool is easy to use. Among the comments given by the participants are that the tool helps users to simplify their use case and test requirement and they are proud to have a tool that can assist a new developer or requirements engineer to validate the requirements. In ease of learning, 78% of the participants agreed that the tool is easy to learn. 20% of the participants were neutral, while only 2% disagreed that the tool is easy to learn. The results indicate that the majority of the participants agreed that the tool is easy to learn. The participants further commented that they need the tool to overcome the difficulties of the learning process so that they can learn aspect of security requirements especially security attributes and validation process at early. The respondents were also requested to state their satisfaction of the tool. In this regard, 83% claimed that they were satisfied with the tool. 16% of the participants were indifference and only 1 % were dissatisfied with the tool. The results indicate the majority of the participants were satisfied with our tool. However, we did not get 100% agreement from the participants that they are satisfied with the tool. Those who were dissatisfied with our tool commented that the tool needs to improve the process to complete testing. Based on the results of the survey, we can conclude that the tool is useful, user friendly and easy to learn. In general, most of the respondents were satisfied with the MobiMEReq tool.

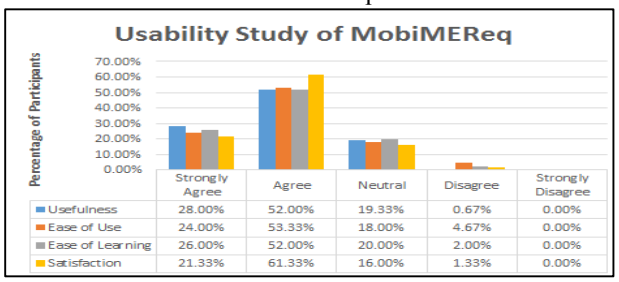


**Fig. 9.** Usability study for capture and validating security requirements on mobile app.

### 5.3 Expert review

We also carried out interviews with two experts in order to get their opinions regarding the usability of our prototype tool. We selected two experts in field of requirements engineering and quality assurance. They have between three to ten years working experience in the information technology (IT) industry. They are from MIMOS and IBM Malaysia and both are experienced in software requirements and test. Expert 1 has 8

years and Expert 2 is 3 years experience. Prior to the interview, we informed them the purpose of the interview and defined the different terminologies and definitions used in our interview questions to ensure the consistency of responses. We provided a brief description of our prototype tool and gave them the access to a link to explore the tool using samples of real-life requirements from their recent projects. Table 8 describes the profile background of the two experts.

Based on the interviews, they agreed that MobiMEReq tool helps to reduce time and human effort in capturing and validating security requirements for mobile apps. They found the tool to be simple, easy to understand and learn. Nevertheless, they provided some constructive comments and feedbacks that are valuable for our research study. Expert 1 (E1) suggested that the tool with mobile apps should be applied for non-enterprise application as the tool can be used as a platform to test single mobile apps to validate user's needs or satisfaction rather than for the organization to access the mobile apps. This may help us to take less elapsed time to implement and validate in a small business application instead of cross of organization. In this case, the validation of security requirement with single test can be focus for whole process of business requirements. E1 also recommended that the security algorithm should be embedded in the tool for better testing execution efficiency. By producing this security algorithm, this may help to protect the privacy of the user's sensitive information and data. Expert 2 (E2) highlighted that the tool is useful for testing mobile apps during eliciting the requirements in terms of functionality and security considering that both RE and Software Developer are not involved at the testing stage in SDLC, in current practice. This tool allows them to perform testing at the early stage of SDLC. E2 felt that the RE can use the tool for demonstrations to clients. He also recommended that the tool could be applied by a Software Developer for testing their work in the back-end process, such as web services when used by the mobile apps.

## 6 Related Work

There have been many methods, approaches, techniques and tools used to validate security requirements for mobile applications. For example, Rhee et al. [22] discussed a methodology to test the Mobile Device Management (MDM) agent. The authors proposed the items and method to identify security requirements, the process and the real world test methods for android devices. Nevertheless, the security of the MDM agent to strengthen the security of the mobile devices needs improvements in validating mobile devices. Farhood et al. [23] proposed a data-centric model to protect all the vulnerabilities to prevent application and malware threats. They proposed a model to ensure the confidentiality, integrity and availability of data stored in mobile devices. However, they did not include validation for security requirement in their study. Gilbert et al. [24] applied the App Inspector, an automated security validation system that analyses application and generates report for potential security and privacy violation. The authors described the future need for making a more secured smartphones applications through automated validation. However, testing of the application does not cover validation in all aspects of security requirements. Gautam et al. proposed a novel secure data access

and monitoring framework with data stored on employee's devices. The tool, referred as Concord identifies critical organizational data that have been accessed by the user's access and theft or loss of mobile device using the cryptography format. However, this tool does not provide a proper validation to confirm that the file is secure [25]. They claimed that security concerns have become very important especially when performing financial transactions, hence the assault vectors in this application need to be examined with due diligence [26]. In this respect, both the users and law enforcement agencies must tackle the issues of mobile security to reduce the risk of criminal misuse. Kamalrudin et al. [27][28] developed a technique and toolset, MaramaAI to support requirements capture and consistency management using EUCs. This tool is supplemented by end-to-end rapid prototyping support. The tool uses EUC patterns to validate requirements consistency, completeness and correctness. However, it focuses on capturing language requirements only, hence the application of this tool in mobile application security requirements is beyond the scope of their work.

## 7 Summary

Security engineers need to validate security requirements for mobile application at an early stage of development. We have developed an automated validation approach and tool support called MobiMEReq for security requirements of mobile apps by adopting the idea of Test Driven Development (TDD) with model-based testing strategy using EUCs and EUIs prototype model. Evaluation of our prototype tool with real security examples and end users shows positive results.

## 8 Acknowledgement

We would like to thank Universiti Teknikal Malaysia Melaka and Sciencefund grant: 01-01-14-SF0106 and also Ministry of Education (MOE), MyBrain15 for support.

## References

1. Schneider, K., Knauss, E., Houmb, S., Islam, S. and Jurjens, J.: Enhancing security requirements engineering by organizational learning. Requirements Engineering 17(1). (2011)
2. Kamalrudin,M. and Grundy,J.: Generating essential user interface prototypes to validate requirements, Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, 564-567, (2011).
3. Paja,E., Dalpiaz, F., Poggianella, M. and Roberti, P. 2012. STS-tool: Socio-technical Security Requirements through social commitments, Proceeding of the Conference 21st IEEE International Requirements Engineering Conference (RE), 331-332, (2012).
4. Yusop, N., Kamalrudin, M., Yusof, M.M., Sidek, S.: Challenges in eliciting security attributes for mobile application development. In Proceeding of the Conference KSII The 7th International Conference on Internet (ICONI), Kuala Lumpur, Malaysia ( 2015)
5. Yahya, S., Kamalrudin, M., Safiah, S., Grundy, J.: Capturing Security Requirements Using Essential Use Cases (EUCs), First Asia Pacific Requirements Engineering Symposium, APRES 2014, April 28-29, 2014, pp. 16-30. Auckland, New Zealand (2014)

6. Paja, E., Dalpiaz, F., Poggianella, M., Roberti, P.: STS-tool: Socio-technical Security Requirements through social commitments. In Proceeding of the Conference 21st IEEE International Requirements Engineering Conference (RE), 331-332, (2012)
7. SANS Institute, Determining the Role of the IA/Security Engineer, InfoSec Reading (2010)
8. Constantine, L. L. and Lockwood, L. A.: Software for use: a practical guide to the models and methods of usage-centered design, Pearson Education (1999)
9. Biddle.R , Noble.J and Tempero, E.: Essential use cases and responsibility in object oriented development. In Proceeding of the 25th Australasian Computer Science Conference. Australian Computer Society, Inc. Chicago (2002), Vol. 24, No. 1, 7-16, (2002)
10. Constantine, L. L. and Lockwood, A. D. L.: Structure and style in use cases for user interface design, in Object modeling and user interface design: designing interactive systems, Addison-Wesley, pp. 245-279, Longman Publishing Co., Inc (2001)
11. Ambler, S.W.: Essential (Low Fidelity) User Interface prototypes. Available from:www.agilemodeling.com/artifacts/essentialUI.htm (2016)
12. Constantine, L.L. and Lockwood, A.D.L.: Usage-centered software engineering: an agile approach to integrating users, user interfaces, and usability into software engineering practice. In Proceeding of 25th International Conference on Software Engineering (ICSE'03) 2003, IEEE Computer Society, Portland, Oregon (2003)
13. Ambler, S.W.: The Object Primer: Agile Model-Driven Development with UML 2.0 (3rd ed.), New York Cambridge University Press (2004)
14. Kamalrudin, M., Grundy, J., Hosking, J.: Tool Support for Essential Use Cases to Better Capture Software Requirements. In Proceeding of IEEE/ACM international conference on Automated software engineering, 327–336, (2010)
15. Kamalrudin, M.: Automated Software Tool Support for Checking the Inconsistency of Requirements .In: 24th IEEE/ACM International Conference on Automated Software Engineering, ASE 2009. IEEE (2009)
16. Kamalrudin, M.: Automated Support for Consistency Management and Validation of Requirements". PhD thesis. The University of Auckland (2011)
17. Yusop, N., Kamalrudin, M. and Sidek, S.: Capturing security requirements of mobile apps using MobiMEReq, 3rd Asia Pacific Conference on Advanced Research, Melbourne, Victoria, Australia (2016)
18. Yusop, N., Kamalrudin, M. and Sidek, S.: Security Requirements Validation For Mobile Apps: A Systematic Literature Review, Jurnal Teknologi(Science & Engineering) 77:33, 123-137, (2015)
19. Kumar, Dr. V., Sujata and Kumar, M.: Test Case Prioritization Using Fault Severity. International Journal of Computer Science and Technology (2010)
20. Novak, V., Perfilieva, I. and Mockor, J.: Mathematical principles of fuzzy logic Dodrecht: Kluwer Academic (1999)
21. Bhasin, H., Gupta, S. and Kathuria, M.: Implementation of regression testing using fuzzy logic. International Journal of Application or Innovation in Engineering and Management, volume 2, issue 4, April 2013, (2013)
22. Rhee, K., Kim, H. and Na, H.Y. 2012. Security Test Methodology for an Agent of a Mobile Device Management System. Int. J. of Security and Its Applications, vol. 6, no.2, (2012)
23. Dezfouli, F.N., Deghantanha, A., Mahmood, R., Sani, N.F.M., and Shamsuddin, S.: A Data-centric Model for Smartphone Security, vol.5, 9-17, (2013)
24. Gilbert, P. and Cun, B. 2011. Vision: Automated Security Validation of Mobile Apps at App Markets. . In Proceeding of the 2nd International Workshop on Mobile Cloud Computing and Services(MCS 2011), 21-26, New York, USA (2011)

25. Singaraju, G., Hoon, B.: Concord: A Secure Mobile Data Authorization Framework for Regulatory Compliance. In Proceeding of the 22nd Large Installation System Administration Conference (LISA '08 ). 91-102, (2008)
26. Ying, L., Dinglong, H., Haiyi, Z. and Rau, P. 2007. Users' Perception of Mobile Information Security. Hacker Journals White Papers. Computer Security Knowledge Base Portal (2007)
27. Kamalrudin, M., Grundy, J., Hosking, J.: Managing Consistency between Textual Requirements. Abstract Interactions and Essential Use Cases. In Proceeding of 2010 IEEE 34th Annual Computer Software and Applications Conference, 327–336, (2010)
28. Kamalrudin, M., Grundy, J. and Hosking, J.: Improving Requirements Quality using Essential Use Case Interaction Patterns. In Proceedings of 2011 International Conference Software Engineering. Honolulu, Hawaii, USA (2011)