

# Using security robustness analysis for early-stage validation of functional security requirements

Mohamed El-Attar · Hezam Akram Abdul-Ghani

Received: 17 December 2013 / Accepted: 24 July 2014 / Published online: 13 August 2014  
© Springer-Verlag London 2014

**Abstract** Security is nowadays an indispensable requirement in software systems. Traditional software engineering processes focus primarily on business requirements, leaving security as an afterthought to be addressed via generic “patched-on” defensive mechanisms. This approach is insufficient, and software systems need to have security functionality engineered within in a similar fashion as ordinary business functional requirements. Functional security requirements need to be elicited, analyzed, specified and validated at the early stages of the development life cycle. If the functional security requirements were not properly validated, then there is a risk of developing a system that is insecure, deeming it unusable. Acceptance testing is an effective technique to validate requirements. However, an ad hoc approach to develop acceptance tests will suffer the omission of important tests. This paper presents a systematic approach to develop executable acceptance tests that is specifically geared for model-based secure software engineering processes. The approach utilizes early-stage artifacts, namely misuse case and domain models, and robustness diagrams. The feasibility of the proposed approach is demonstrated by applying it to a real-world system. The results show that a comprehensive set of security acceptance tests can be developed based upon misuse case models for early-stage validation of functional security requirements.

**Keywords** Misuse case model · Security robustness analysis · Acceptance tests · Functional security requirements

## 1 Introduction

Nowadays, software systems are rarely developed to operate in a stand-alone mode. All software systems are connected to other systems that may inflict harm, and therefore, defensive mechanisms need to be in place in order to mitigate such threats. It is also unrealistic to assume that human users of software systems will always intend to use it in a legitimate manner. Misuse of software systems by humans must also be addressed. Traditional methods of software development focus on the implementation of business-related functional requirements while addressing security issues toward the end of the development process. Security is addressed by supplementing the end system with defensive mechanisms such as firewalls, cryptography and IDS (intrusion detection system). Research evidence has proven that such approaches to address security-related concerns are insufficient and will likely cause costly reworks in addition to any intangible consequences caused by a security breach [1]. To avoid these costly reworks, security concerns need to be addressed as early as the requirements engineering phase. To this end, secure software engineering has recently become a very active area of research.

Functional requirements validation is a crucial activity in any software development process. Overlooking requirements validation can lead to the development of the incorrect system (a system that does not satisfy its functional requirements). Similarly, functional *security* requirements need to be validated. Failure to validate

---

M. El-Attar (✉) · H. A. Abdul-Ghani  
Information and Computer Science Department,  
King Fahd University of Petroleum and Minerals,  
P.O. Box 5066, Dhahran 31261, Kingdom of Saudi Arabia  
e-mail: melattar@kfupm.edu.sa

H. A. Abdul-Ghani  
e-mail: akramhezam2008@gmail.com

functional security requirements can lead to the development of an insecure system. A mis/use case model can be used as a basis for requirements validation. However, the requirements validation process can be made more rigorous using acceptance testing [2]. Acceptance testing in the requirements engineering phase can be very beneficial as it is cost effective. Acceptance testing provides a new viewpoint for customers to validate a system's requirements through a set of tests (i.e., understanding through examples of use). In a secure software engineering process, acceptance testing is equally beneficial since it can be used as a basis for early-stage validation of functional security requirements. The development of these tests and dry-running their expected outputs also provides developers with a more accurate understanding of a system's expected behavior. The developed acceptance tests can be used to define acceptable external quality, to refine the requirements specifications and to track a project's progress. The tests can be created using simple syntax which allows them to be developed quickly while being understandable to non-technical stakeholders. The syntax used to create the acceptance tests, although simple, is sufficient to make them machine executable. The literature constantly urges for increased customer involvement during requirements construction and system evaluation, and acceptance testing partially fulfills this perceived need [3].

The current state of practice finds acceptance testing mostly deployed in agile processes and underutilized in the more formal, model-driven, processes. Since acceptance tests are developed based on requirements artifacts, therefore they are often constructed from user stories [4]. It will be advantageous to reap the benefits of acceptance testing in large-scale development projects. However, large-scale development projects do not use user stories. Large-scale development projects deploy a more rigorous, model-oriented, development process such as the V-Model. The UML is commonly used in such large-scale projects. In the requirements engineering process, functional requirements are elicited, communicated and modeled as use cases. Although use case modeling is a very popular technique in industry, it does not support the elicitation and specification of functional security requirements. To counter this deficit, use case modeling has been extended by Sindre and Opdahl [5] to account for functional security requirements. The extended modeling technique is named misuse case modeling [5]. The misuse case modeling technique has emerged during the past decade as a promising technique to elicit and model functional security requirements [5–8]. Misuse cases define improper usage scenarios of a system by its external entities that may lead to harmful consequences in a very similar way that use cases describe legitimate usage instances. The literature has already reported a number of successful industrial experiences with misuse case

modeling [9, 10]. In order to reap the benefits of acceptance testing in large-scale projects and validate functional security requirements, we propose an approach to develop acceptance tests from misuse cases. The approach proposed in this paper provides a systematic method to develop a comprehensive set of executable security acceptance tests using artifacts already available during the requirements phase. The artifacts used are a misuse case model and a domain model. The proposed approach uses these artifacts to perform security robustness analysis to develop security robustness diagrams. It is important to note that our proposed approach is not intended to replace or improve upon any other approaches that develop acceptance tests. In fact, we recommend using our approach in parallel with other requirements validation techniques.

The remainder of this paper is structured as follows: Sect. 2 provides a brief introduction to misuse case modeling. Section 3 discusses related research works. In Sect. 4, the proposed methodology is presented. Automation support is discussed in Sect. 5. Section 6 presents a case study pertaining to a FSC subsystem. Finally, Sect. 6 concludes and suggests future work.

## 2 Background

The misuse case modeling technique was introduced by Sindre and Opdahl in 2000 [5] as an extension to use case modeling. This extension was proposed since the use case modeling notation and its semantics do not support the elicitation and modeling of functional security requirements. The misuse case modeling notation and its semantics were purposely designed with great resemblance to the use case modeling notation. The main contribution of misuse case modeling is the introduction of two new concepts, namely misuse cases and misusers. The notation and semantics of misuse cases and misusers are in line with the definition of use cases and actors, respectively. Misuse cases and misusers are defined in [5] as follows:

*Misuse Case: "A Sequence of actions, including variants, that a system or other entity can perform, interacting with misusers of the entity and causing harm to some stakeholder if the sequence is allowed to complete" [5].*

*Misuser: "An actor that initiates misuse cases, either intentionally or inadvertently" [5].*

Misuse cases may share the same relationships between them as do use cases, namely the *include*, *extend* and *generalization* relationships. Misusers may only share a *generalization* relationship between them as do actors. Misuse cases and misusers are depicted as ovals and stickman figures, respectively, to signify their semantic

resemblance to use cases and actors. However, misuse cases and misusers are depicted with inverted colors to signify that they are the inverse of use cases and actors, respectively. Misusers are linked with misuse cases using only the *association* relationship, which is also the only relationship that can exist between actors and use cases. Misuse case modeling introduces two new relationships: the *threatens* and *mitigates* relationships. A *threatens* relationship may only be directed from misuse cases to use cases. The *threatens* relationship states that the given misuse case threatens the security of the system when the given use case is being performed. The *mitigates* relationship may only be directed from use cases to misuse cases. The *mitigates* relationship states that the given use case is performed to mitigate against the threat posed by the given misuse case [5].

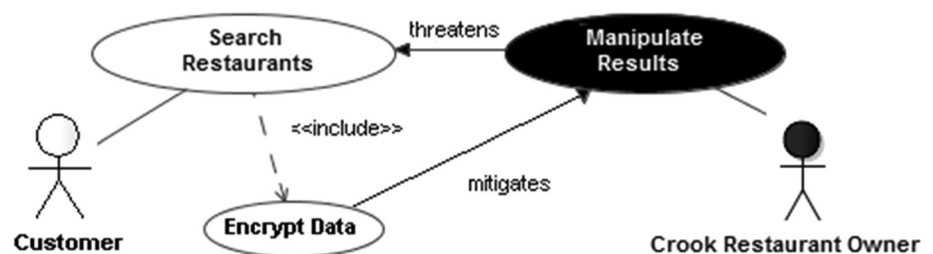
Misuse cases depicted in the diagram are supplemented with textual descriptions. The textual descriptions explain in detail the threatening behavior of each misuse case. There are various templates proposed to describe misuse cases [11, 12]. The proposed templates are often based on popular use case templates such as in [13, 14]. The behavior is therefore described at the interaction level between the underlying system and other entities external to it. The actual behavior of each use case and misuse case is obtained from the textual descriptions while the diagram acts as a visual summary to all entities involved. Figure 1 shows an example of a misuse case diagram.

The misuse case diagram shown in Fig. 1 is concerned with a restaurant search feature available to customers. The feature is described by the “Search Restaurants” use case. A crook restaurant owner intercepts the search request and manipulates the results, as modeled by the “Manipulate Results” misuse case, most likely to promote their own restaurant unfairly. To defend against this threat, a new use case named “Encrypt Data” is introduced to *mitigate* the “Manipulate Results” misuse case by encrypting the search request.

### 3 Exploiting use cases to derive acceptance tests

A number of research works have proposed various methods to develop tests based on use case descriptions

**Fig. 1** A misuse case diagram example



[15–18]. Unlike the approach proposed in this paper which produces *acceptance* tests, the type of tests created in these works is *system* tests [15–18]. System testing is an essential component of the overall verification and validation effort as stated in the V-Model. However, there are differences between acceptance and system testing and thus neither can substitute for the other. As a prelude to comparing our approach with the other works, the distinction between acceptance and system testing is identified. Table 1 compares acceptance and system testing with respect to their purpose and their intended users.

The purpose and properties of acceptance testing, as outlined in Table 1, suggest that any technique aimed at developing acceptance tests should ideally satisfy the following criteria:

*Criteria 1* Low technical difficulty to be used by customers and business analysts (BAs) as they will be highly involved.

*Criteria 2* Applicable in the early phases of the development life cycle to be useful for validation.

*Criteria 3* Bridges the gap between the analysis and design phases.

*Criteria 4* Produces executable tests.

*Criteria 5* Produces reusable tests.

*Criteria 6* Produces tests that cover inter-use case relationships, namely the *include* and *extend* relationships.

*Criteria 7* Produces tests that validate and verify functional *security* requirements.

*Criteria 8* Describes in detailed how to be systematically applied.

The TOTEM (Testing Object-Oriented systems with the Unified Modeling Language) methodology utilizes various analysis artifacts, such as use case models, sequence, collaboration and class diagrams to develop test cases [16]. Sequence, collaboration and class diagrams are usually unavailable until after the completion of the design phase (criterion-2). The TOTEM methodology prescribes that analysis artifacts need to be heavily augmented with OCL (Object Constraint Language) expressions. The technique introduced in [17] was also based on extending use cases with contracts to facilitate test case generation. It is safe to assume that learning and using OCL effectively is

**Table 1** Acceptance testing vs. system testing

	Acceptance testing	System testing
What and When?	<p><i>Prior to development</i></p> <p>Acceptance test development is a validation process in whereby the tests are created and their results dry-run for the purpose of ensuring that the “correct” system will be built. There is multitude of sources for acceptance tests. In large-scale projects, tests are commonly derived from use cases and domain models</p> <p><i>After development</i></p> <p>Acceptance testing here is a verification process used to show the customer that the system satisfies the agreed upon requirements. Acceptance testing is used to determine that the software developed is properly operating on-site. Moreover, it is used as checklist and a basis for contract satisfaction</p>	<p>System testing is a verification process the exercises the system to determine whether it actually produces the expected results based on given input. System testing is performed before delivery to observe the system behavior as a whole and to detect and fix bugs. System testing is therefore conducted only after the SUT (system under test) has been developed. System tests are usually derived from acceptance tests and other system design artifacts</p>
Who?	<p><i>Customers, end users and business analysts (or requirements engineers)</i> ideally will collaborate during the requirements engineering phase to create user acceptance tests. Customers and end users are crucial aspect since they possess valuable about the problem domain and the required functionalities. Business analysts apply analytical techniques to derive tests to determine whether the requirements are complete, consistent and correct</p>	<p><i>Software developers</i> will ideally create system tests to guide their development activities. <i>Software testers</i> will use the developed system tests to check for bugs</p>

an advanced skill beyond what can be expected from a customer or a BA (criterion-1). According to the duties and skills set of BAs outlined by BABOK (Business Analysts Body of Knowledge) [19], which is a well-recognized official reference used by BAs to attain their BA certification, it can be deduced that a BA cannot be expected to perform any activities that can be considered technically highly complex. In [18], an approach named the SCENT-Method was introduced. The SCENT-Method is concerned with systematically constructing statecharts from use case scenarios. The statecharts are detailed with pre- and post-conditions, data ranges, data values and non-functional requirements. Users of the SCENT-Method need to cross-check the statecharts for any inconsistencies, incorrectness and incompleteness. Once again, this approach is technically too demanding to be performed effectively by a customer or a BA (criterion-1). According to BABOK, the skill set required by the approach presented in this paper should be possessed by BAs (criterion-1) [19]. The only exception is the skill of performing robustness analysis. However, in previous work [20], it was empirically validated that BAs can effectively perform robustness analysis using a small learning curve and without the need for in-depth knowledge of object-oriented concepts.

None of the previously mentioned approaches produce executable acceptance tests (criterion-4), which in turn hinders their reusability of the tests created (criterion-5). The approach presented in [15] can be used to derive executable test cases from UML diagrams. The limitation of this approach is that it requires detailed sequence and

communication diagrams to be available. These are detail design artifacts that are ideally expected to be constructed and communicated by system designers not customers and BAs (criteria-1 → 3). Therefore, the requirements of the approach presented in [15] prevent it from being performed during the early phases of the development process (criterion-2). An approach developed in previous work [21] satisfies criteria-1 → 5, but not criteria-6 and 7. The approach presented in this paper was specifically designed to account for functional security requirements and inter-use case relationships, in addition to satisfying criteria-1 → 5. There are important characteristics about the approach presented in this paper which differentiates it from the approach in [21]. Firstly, the approach presented in [21] only considered use cases, not misuse cases. Although misuse cases are semantically similar to use cases, their execution *interferes* with the execution of a use case. This means that misuse cases are performed in parallel with use cases. Misuse case behavior does not simply *plug-into* the behavior of a use case as is the case of a function calling another function. In fact, this characteristic of misuse case modeling motivated the development of mal-activity diagrams in order to model and communicate misuse behavior more accurately [22]. The behavioral characteristics of misuse cases significantly compound their analysis which is required to produce acceptance tests. Secondly, the approach presented in [21] does not account for relationships between use cases such as the *include* and *extend* relationship. The approach presented in this paper accounts for such relationships, in addition to relationships

between misuse cases and use cases, namely the *threatens* and *mitigates* relationships. The approach presented in [23] is similar to [21] with the exception that it accounts for relationships between use cases. The approach in [23] does not account for security aspects (criterion-7). The approach in [23] does not utilize robustness diagrams, and hence, it does not bridge the gap between the analysis and design phases (criterion-3). The approach presented by [24] allows for acceptance testing but as a verification tool not a validation tool. This means that the approach by [24] would ideally be used toward the end of a V-Model development process. The approach by [24] performs acceptance testing by aggregating unit tests into integration tests, then integration tests into system and acceptance tests. This requires a great deal of design and perhaps coding to be performed upfront. Therefore, the approach presented by [24] does not satisfy criterion-2. The approach presented by [24] also does not account for relationships between use cases, and it does not consider security aspects specifically with misuse cases. In [25], the authors present one of the earliest documented use case-based acceptance testing approaches. The approach was described at a very abstract level, and it also does not account for security aspects. Apart from [16, 17], no other approach presented a detailed and systematic approach to their methodologies. Most approaches were presented at a high level that leaves its users with the challenge of how to apply them precisely and effectively, including our previous work [21]. A summary of the above-mentioned criteria satisfaction by the various approaches for test generation is presented in Table 2.

## 4 Methodology

In this section, we will explain the proposed methodology that will yield executable acceptance tests to validate functional security requirements stated in misuse case models. The proposed methodology consists of three main phases. The main phases are briefly described below, and an overview of the overall methodology is shown in Fig. 2.

Tasks that are fully or partially tool supported are annotated with the label “auto.” A detailed of the automation support provided is presented in Sect. 5. The main phases of the proposed methodology are described in greater detail in Sects. 4.1, 4.2 and 4.3.

*Phase 1* Develop security acceptance tests at a high level based on each set of interrelated use and misuse cases (Fig. 3). Use and misuse cases are considered interrelated when they form a web of associations that consists of a usage scenario; behavior that threatens the system when performing this usage scenario; and mitigation behavior. The high-level security acceptance tests (HLSATs) are used to draw quick feedback from stakeholders to validate the functional security requirement. The other purpose of the high-level acceptance tests to determine the necessary inputs or triggers of the use and misuse cases.

*Phase 2* Perform security robustness analysis to create and utilize objects that realizes the behavior of the use and misuse cases. A separate security robustness diagram will be created to model the objects representing a set of interrelated use and misuse cases.

*Phase 3* For each security robustness diagram, its object-level information is used to realize the high-level security acceptance tests previously developed in phase 1 by transforming them into an executable form.

*Phase 4* This phase occurs post-development. In this phase, the executable security acceptance tests created are used to test the system in order to verify that it is functioning as expected in case of misuse.

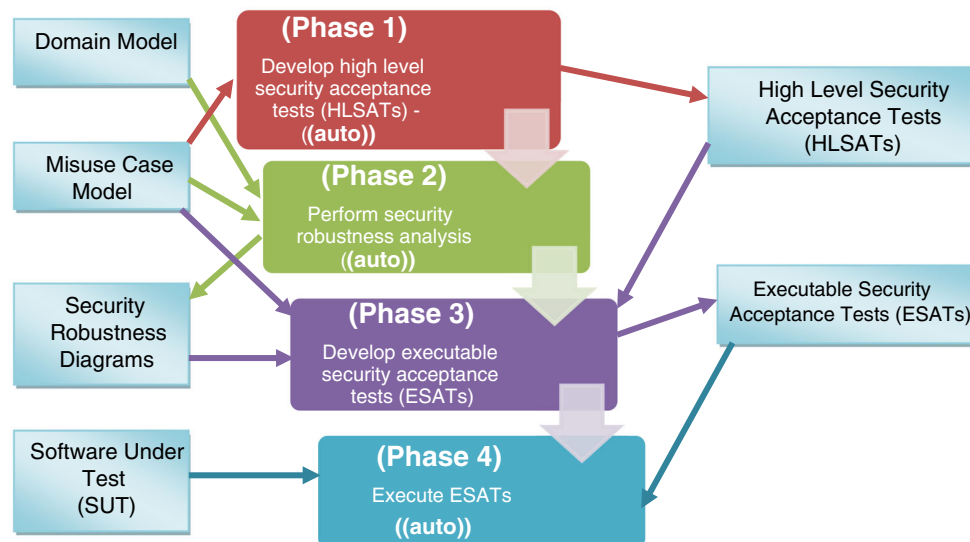
### 4.1 Phase 1: Developing high-level security acceptance tests

The narratives of use and misuse cases are used as basis for developing informal and abstract-level descriptions of security acceptance tests (HLSATs). HLSATs are developed to disconnect the process of identifying acceptance tests from any technical details such as that concerned with the syntax of a programming language. As such, the user of

**Table 2** Comparison of techniques presented in the literature

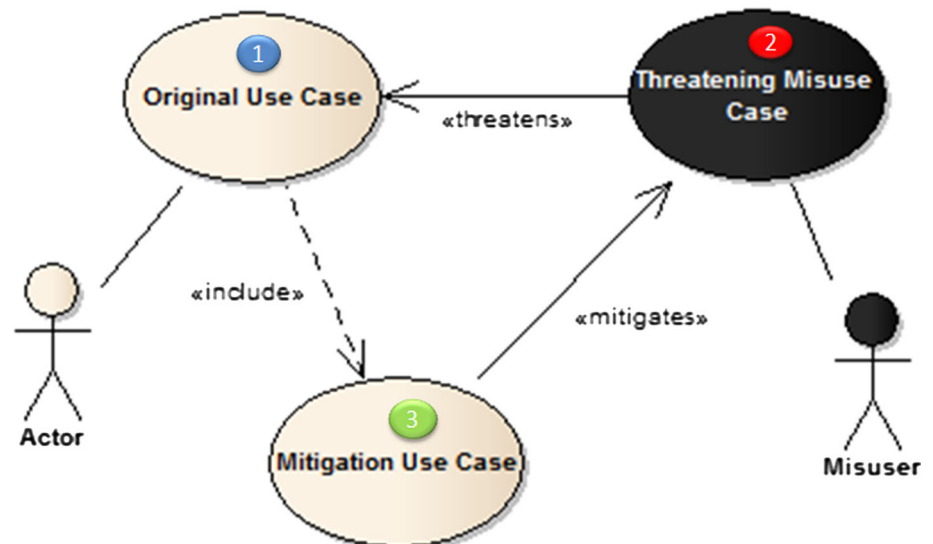
Criterion	[16]	[17]	[18]	[15]	[24]	[25]	[23]	[21]	This Paper
1						✓	✓	✓	✓
2		✓				✓	✓	✓	✓
3			✓		✓			✓	✓
4				✓	✓	✓	✓	✓	✓
5				✓	✓	✓	✓	✓	✓
6	✓	✓					✓		✓
7									✓
8	✓	✓							✓





**Fig. 2** An overview of the proposed methodology

**Fig. 3** Usage patterns in misuse case models



the proposed technique will be able to focus on determining the essential set of acceptance tests without being distracted by syntactical details. HLSATs are developed by determining sets of use cases, misuse cases that *threatens* them and their mitigation use cases. For each set, their narratives are analyzed to systematically create the necessary acceptance tests. The acceptance tests developed should cover the various usage scenarios. As is the case with any type of test, acceptance tests are comprised of inputs and expected outputs. Input can be in the form of data or a series of functional calls. Tests are evaluated by checking the system's resulting output or final state. In order to determine this essential information, the user of the proposed technique asks three key questions:

*Question 1: What are the usage scenarios that span the set of use cases, their misuse cases and their mitigation use cases?*

Usage scenarios are not confined to just one mis/use case. Misuse scenarios and the counter mitigation behavior are described across a pattern of mis/use cases. The pattern includes ordinary use cases that contain business-related behavior, misuse cases that execute harmful behavior and mitigation use cases that describe the behavior necessary to mitigate the threatening behavior. Ideally, the threatened use case will *include* the mitigation use case to call upon its mitigation behavior. This pattern is shown in Fig. 3. The usage patterns detected will lead to the identification of usage

scenarios in the form of *single flows*. Here, *single flows* mean flows that do not contain alternate branching, but flows that relate to one usage scenario. This process is completely automated as discussed later in Sect. 4.

Assuming a faculty search committee (FSC) system that is used to process faculty employment applications, which will be used as a running explanatory example throughout this paper. Figure 4 presents an excerpt of the complete misuse case diagram. Figure 4 presents an instance of a usage pattern as shown in Fig. 3.

This misuse case is initiated by a malicious applicant by providing a username that does not belong to them (unless the misuser is an insider). The system then displays the list of active applications. The malicious applicant then provides the name or ID of a particular application which is then retrieved and displayed by the “Application Viewer.” The malicious applicant then executes a remove function to remove that retrieved application. The expected mitigation behavior is then triggered by the system by adding the

removed application to a list removed applications. The list contains the removed applications in addition to the users who executed the remove function. Figure 5 presents a shortened version of the use and misuse case textual descriptions. For brevity and simplicity purposes, only the “basic flow” of all misuse and use cases is shown and will be analyzed in details.

The use and misuse case descriptions are transformed into mal-activity diagrams [22]. Mal-activity diagrams use a similar notation to that of UML activity diagrams but with an extended notation to represent misuse. The transformation process is automated using a model transformation tool developed in previous work [26] (more details about automation support in Sect. 5). The mal-activity diagram generated contains all flows that are embedded in a given usage pattern. The creation of HLSATs requires each flow to be considered separately. Therefore, the mal-activity diagram generated is then dissected into separate flows. This task is also automated (Sect. 5). The following

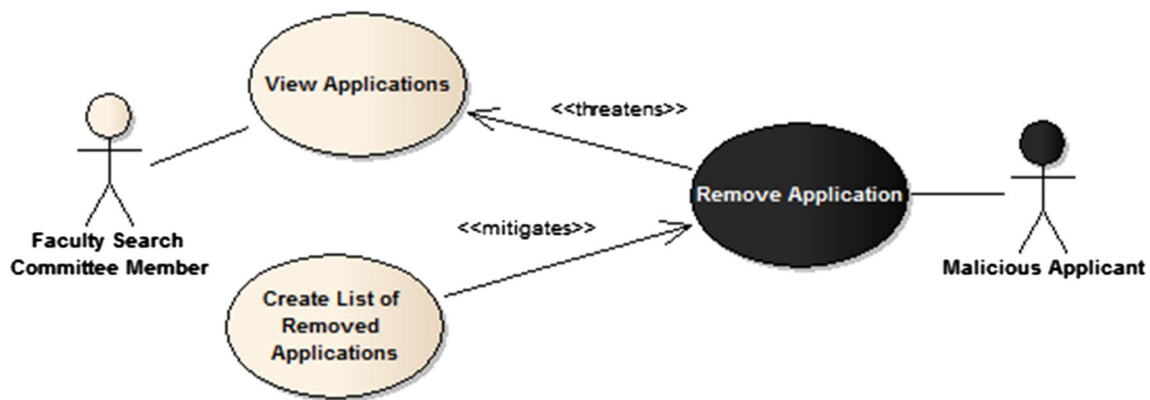


Fig. 4 Instance of a usage pattern from the FSC subsystem

Fig. 5 The textual descriptions of the involved use and misuse cases

View Applications	Remove Application	Create List of Removed Applications
<p><b>Preconditions:</b> FSC subsystem must be running</p> <p><b>Basic Flow:</b></p> <ol style="list-style-type: none"> <li>1. Faculty Search Committee member enters user name and password to logon.</li> <li>2. The system displays all active applications.</li> <li>3. The system displays all removed applications.</li> <li>4. The Faculty Search Committee member selects an applicant to in more detail.</li> <li>5. The system displays the selected application in more detail</li> </ol>	<p><b>Basic Flow:</b></p> <p>Before <b>Step 1</b> in <b>Basic Flow</b> of <b>View Applications</b></p> <ol style="list-style-type: none"> <li>1. Malicious applicant gains access to a legitimate user’s credentials</li> <li>2. Malicious applicant enters the user credentials to logon.</li> </ol> <p>After <b>Step 5</b> in <b>Basic Flow</b> of <b>View Applications</b></p> <ol style="list-style-type: none"> <li>3. Malicious application removes the displayed application</li> </ol>	<p><b>Basic Flow:</b></p> <p>After <b>Step 3</b> in <b>Basic Flow</b> of <b>Remove Application</b></p> <ol style="list-style-type: none"> <li>1. Once an application is removed the system stores the application in a list of removed applications.</li> <li>2. Resumes at <b>Step 2</b> of <b>Basic Flow</b> of <b>View Applications</b></li> </ol>

is the mal-activity diagram of the usage pattern from the use and misuse cases shown in Fig. 6, which shows the single-flow usage scenario that pertains to only the basic flows of the use case misuse cases.

*Question 2: What are the input data or actions that triggers and perform the usage scenarios?*

Upon identifying the usage scenarios from Q1, it is required to identify the needed inputs, triggers and preconditions. Inputs and triggers are usually provided by actors, misusers, use cases, misuse cases or other systems. The preconditions are used to determine the requisite system state to establish the proper testing environment. This essential information can be obtained by examining the textual descriptions of mis/use cases, actors, misusers and the domain model. Ideally, the required information should be available within the textual descriptions of the mis/use cases; otherwise, the descriptions are likely incomplete. Input can be provided throughout the execution of a scenario. Therefore, it is necessary to perform a *step-by-step* examination of the textual descriptions in order to determine what input is needed and when it should be provided. The required

data inputs should be identified within the classes of the domain model as attributes. Triggers should be identified within the classes of the domain model as operations. The use of Functional Grammar can guide this task. Functional Grammar is a general theory concerning the grammatical organization of natural languages [27]. Functional Grammar can be used as a means to formalize and structure natural language specifications in use and misuse case descriptions (which at this point are dissected into various usage scenarios). Once again, Functional Grammar is used to identify candidates for the usage scenario inputs. However, caution should be used when using Functional Grammar as it may yield many false positives, i.e., nouns that are not inputs, also some inputs may not be readily identified by tool support. Therefore, manual analysis of the narrative is necessary.

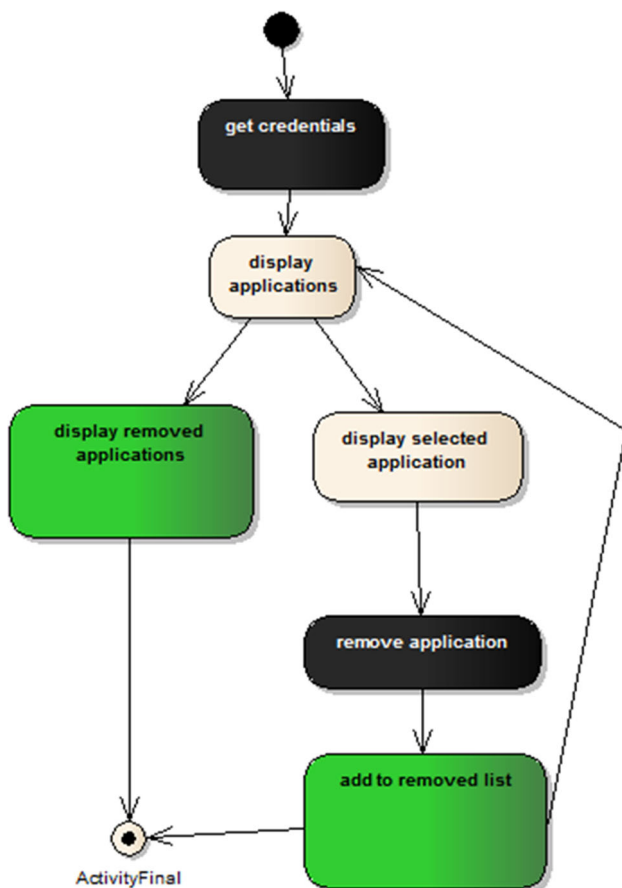
Revisiting the running example of the FSC subsystem, we find that the only precondition is that the FSC subsystem needs to be running. Moreover, we find that the usage scenario requires a username, password and the ID of the application to be removed as input as shown in Fig. 7. The domain model of the FSC subsystem is shown in Fig. 17.

*Question 3: What are the expected output values (or system state) for each usage scenario?*

The success criterion of a usage scenario is ideally evaluated by the output values the system produces or the system final state. Similar to Q2, output can be provided throughout the execution of a scenario. Therefore, it is necessary to perform a *step-by-step* examination of the textual descriptions in order to determine what output is produced and when it will be produced. Evaluating the system final state is naturally performed upon the completion of the usage scenario. Similar to Q2, The output data should be within the classes of the domain model as attributes. Once again, Functional Grammar can be used to systematically analyze sentences to detect nouns that can be candidates for outputs. Once again, Functional Grammar should only be used as an aid to the user. The user should not be fully reliant on the tool support for this task, and manual analysis is still considered necessary.

Revisiting the running example of the FSC system, we find that the only output of this usage scenario is that an application is removed and the removed application is then displayed as shown in Fig. 7.

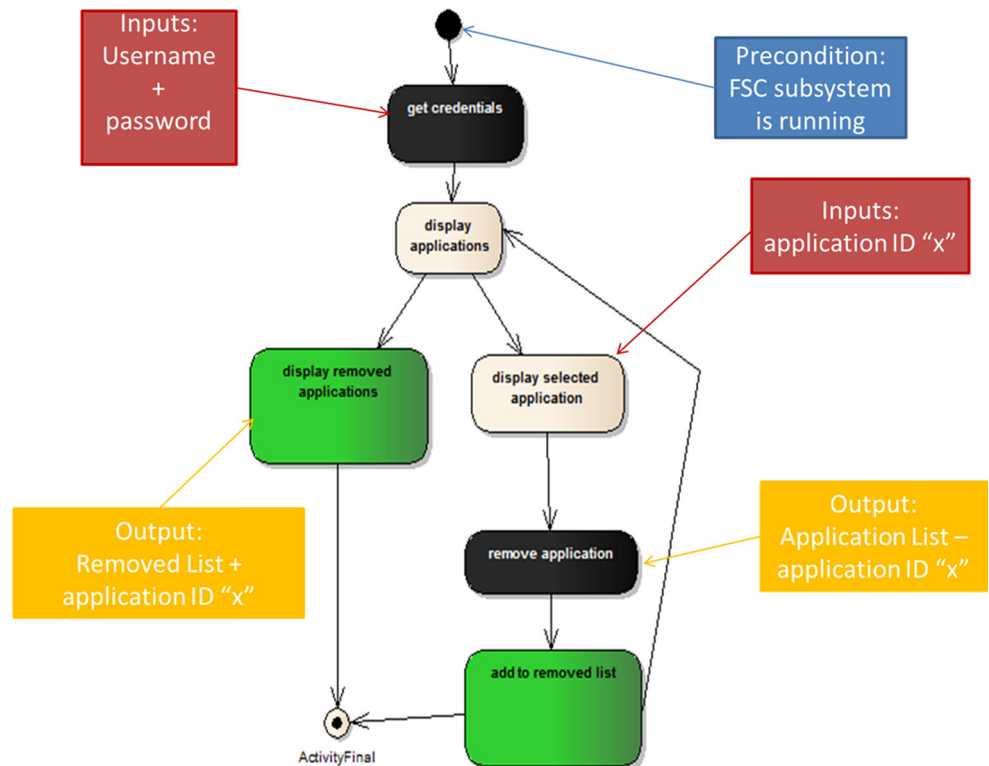
Completing Q3 above will result in the creation of a HLSAT. There will a one-to-one correspondence between each usage scenario and HLSAT. There may be cases where the above three questions are difficult to answer. This is usually symptomatic of a low-quality misuse case



**Fig. 6** Usage scenario of the basic flow of the use and misuse cases



**Fig. 7** Inputs and outputs for the given usage scenario



model. Such misuse case models describe behavior that is ambiguous, too abstract or incomplete. Performing security robustness analysis (Phase 2) will attempt to remedy this problem and improve the quality of the misuse case model. However, if security robustness analysis fails, then the misuse case model used is of very low quality and should be improved beforehand.

For the FSC subsystem, the HLSAT created for the basic flows of the usage patterns shown in Fig. 4 is presented below in Table 3.

#### 4.2 Phase 2: Performing security robustness analysis

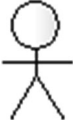



Security robustness analysis was introduced in earlier work [28]. Security robustness analysis is an extension to the robustness analysis technique [29]. Robustness analysis has a dual purpose: (a) it is used to complete and remove inconsistencies between the use case and domain models and making them both more complete, thus making them more *robust*, and (b) its outcome is a robustness diagram which utilizes objects from the domain model required to realize scenarios described in use cases, thus reducing the gap between the analysis and design phases. The source of the objects used is either the domain model or introduced by the modeler. In case of new objects being introduced during security robustness analysis, these objects are added to the domain modeling, making it more detailed.

**Table 3** The HLSAT for the usage scenario involving the “Remove Application” misuse case

Test ID	Description	Expected Results
Remove Application— <i>Basic Flow</i>	<i>Precondition:</i> Run FSC subsystem	<i>Output:</i> The selected application is removed
	<i>Input:</i> Username	<i>Output:</i> The selected application is added to the list of removed applications displayed
	<i>Input:</i> Password	
	<i>Input:</i> ID of application to be removed	

It is important to note that robustness analysis is not intended to yield a final design although the robustness diagrams produced can be refined into a final design if the design team wishes to do so. The main purpose of robustness analysis is to brainstorm and explore different solutions to the problem domain without committing to any particular design prematurely and becoming overly concerned with design details and conforming to a wide variety of syntax rules. Therefore, the robustness diagrams modeling notation is purposely designed to be relatively very simple. The notational set contains four main constructs shown in Table 4. Any two entities can be linked

**Table 4** Robustness diagram objects

Entity	Symbol	Concept
Actors		Similar concept to an actor in use case diagrams
Boundary		Actors communicate with the system using boundary objects
Entity		Similar concept to an entity in a conceptual model
Control		Undertakes logical tasks using boundary and entity objects

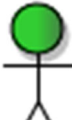







together using a solid line to loosely state that they are associated.

The robustness analysis technique is only applicable to regular use cases, and it only models regular business-related functional behavior. The notational set of robustness diagrams therefore does not support the modeling of security aspects, such as those described in a misuse case. To overcome this drawback, security robustness analysis was introduced as an extension to robustness diagrams [28]. Security robustness analysis yields security robustness diagrams which introduce two new sets of objects that share similar notation to robustness diagrams (Table 4). The first set of objects is used to model the realization of misuse and threatening behavior as described in a misuse case (see Table 5 right). This set of objects is colored black to symbolize their negation property, in-line with misuse cases. The second set of objects is used to model the realization of the mitigation behavior as described by a use case the *mitigates* a misuse case (see Table 5 left), which are colored green. This second set of objects was introduced since it is important to make the distinction between objects representing business-related behavior and objects that were introduced to mitigate against threats.

An initial step toward creating a security robustness diagram is to reuse the activities that were previously derived in the form of single flows (usage scenarios). The activities in the usage scenarios provide suitable candidates for control objects. For the FSC subsystem, the control objects reused are shown in Fig. 8. The control objects have a one-to-one mapping with the activities present in the mal-activity diagram shown in Fig. 6.

The security robustness diagram is developed by analyzing the text of the relevant use and misuse cases. A set of objects are introduced based on the analysis of the text. Analysis of the relevant use and misuse cases narrative should be performed in four phases. For this task, the usage

**Table 5** Extended notation of security robustness diagrams

Entity	Symbol	Entity	Symbol
Mitigator		Misuser	
Mitigation boundary		Misuse boundary	
Mitigation control		Misuse control	
Mitigation entity		Misuse entity	

scenarios derived in the previous phase should be used. In the first phase, the narrative in the usage scenarios pertaining to use cases describing the ordinary business rules is analyzed and the set of regular objects and actors are created, using the notation shown in Table 3. Given that the control objects are derived from **Phase 2**, the focus in this task is to determine the boundary and entity objects that will be used by the regular control objects. Preexisting objects in the domain model should be used to develop this initial diagram whenever possible. The information present in the already developed HLSATs should also be used to guide this task. For the FSC subsystem, it can be determined that there are two types of legitimate actors that can access this functionality: (a) the faculty search committee member and (b) faculty search committee chairman actors. Therefore, two actor objects are created accordingly. It can also be determined from the narrative and the domain model that the “Application Viewer” is the only interface for the actors. As such, a boundary object is created called “Application Viewer.” In order to process the authentication, a user credential is identified from the list of users. Accordingly, two entity objects are created. The first is called “Users” which represent the list of users, while the other will be called “User” which contains information about the active user. It can also be determined that an application is retrieved from list of applications. Similarly, two more entity objects are created; the first is called “Applications List” to represent the array of applications, while the other is called “Application” which represents the active application. The list of newly created regular actors, boundary and entity objects is shown in Fig. 9.

In the second phase, the narrative in the usage scenarios pertaining to misuse cases describing the misuse behavior is analyzed and a set of misuse objects and misusers are created (black graphical notation). Relevant misuse cases



Fig. 8 Control objects

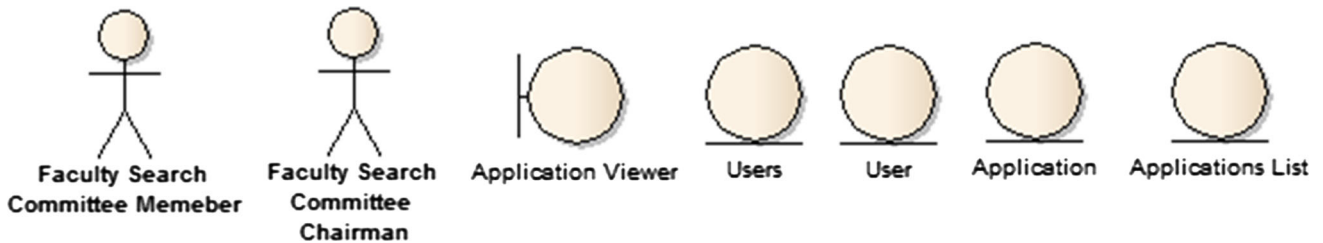


Fig. 9 Actors, boundary and entity objects



Fig. 10 The misuser

should be identified from the misuse case diagram. Relevant misuse cases are the ones that have a *threatens* relationship directed toward the given use case. When analyzing the text of the misuse cases, it is important to determine the points in the functional behavior of the regular use case where the misuse behavior is expected to happen. As such, the robustness diagram initially developed is amended by adding the misuse objects and misusers within the flow of the regular objects and actors. At this point, the robustness diagram only contains objects that realize the business rules, misuse objects, actors and misusers (all graphical entities at this point are either white or black). For the FSC subsystem, it can be determined that there is only one misuser role “Malicious Applicant” and a misuser object is accordingly created with the same name. The misuser does not use new interfaces and does not introduce new entity information, and therefore, no new misuse boundary or entity objects are created. The newly created misuser is shown in Fig. 10.

In the third phase, the narrative of the mitigating use cases is analyzed and a set of mitigation objects and actors are created (green graphical notation). Relevant mitigation use cases can be identified as the ones that have a *mitigates*



Fig. 11 The mitigation entity

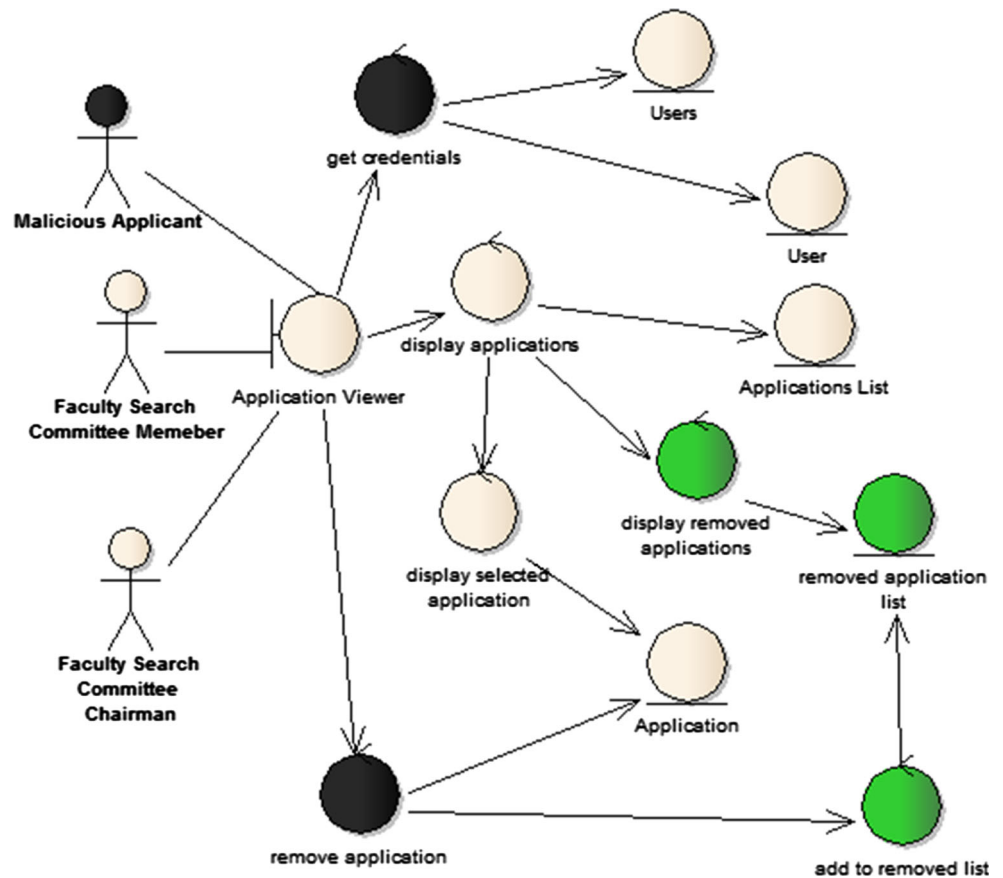
relationship directed toward misuse cases that have a *threatens* relationship directed toward to the original use case. For the FSC subsystem, mitigation control required the new creation of a list that stores removed applications. Therefore, only one mitigation entity object is created and named “removed application list” as shown in Fig. 11.

The final phase of developing a secure robustness diagram is to create association relationships that together *play-out* the behavior described in the usage scenarios. To perform this task, association relationships are created according to the logical flow described in the usage scenarios. A sanity check should be performed upon creating the secure robustness diagram to ensure its correctness and completeness. The sanity check is performed by tracing the behavior described in the usage scenarios through the linked objects. For the FSC subsystem, the secure robustness diagram is developed as shown in Fig. 12.

#### 4.3 Phases 3 and 4: Developing and executing executable security acceptance tests (ESATs)

This final phase leads to the ultimate goal of developing ESATs. For this phase, developers can use any automated framework to develop and execute the security acceptance

**Fig. 12** The security robustness diagram for the “Remove Application” misuse case



tests. There exist many tools that provide automation support for this phase, such as FIT/FITnesse [30] and Selenium [31]. Although our approach is independent of any implementation solution, we utilize the FIT/FITnesse syntax, since it is arguably the most commonly used framework for developing and executing acceptance tests. Using the FIT/FITnesse framework requires acceptance tests to be developed in a tabular form, known as *fixtures*. FIT/FITnesse provides three types of *fixtures*: (a) Action-Fixture, (b) RowFixture and (c) ColumnFixture. Each fixture type is used to develop different categories of test cases.

An ActionFixture can be used to develop scenario-based tests that includes multiple steps. ActionFixtures are especially useful in transaction-based functionalities where frequent interactions with a user are expected. The rows in an ActionFixtures are executed sequentially where each row (except the fixture header) is considered an *action*. Each row consists of a numbers of fields. The first field of each row contains one of four commands that indicate the type of action to be performed. Table 6 provides a brief explanation of each command. The subsequent fields are used to include the necessary information to execute the command. A RowFixture is used to check sets of data. The header of a RowFixture indicates the name of the data

**Table 6** Keywords used in ActionFixtures

Command	Purpose
Start	Starts the given application. It is useful to execute this command to ensure the initial state of the system
Enter	This command is used to create input. The following field contains the name of input to be entered, which is followed by another field containing the actual data value
Press	This command is used to execute functions in the given application. The following field contains the name of the function to be executed. Note that it is often the case that a function is executed by performing a GUI operation, such as a push of a button
Check	The purpose of this command is to evaluate whether the output rendered by the application is the same as the expected output. The following field contains the name of the data to be evaluated, which is followed by another field containing the expected data value

structure to be evaluated. The remaining rows include the expected data elements. ColumnFixtures are used for formula evaluations where input is provided to a function and its output is evaluated. ColumnFixtures are useful to evaluate functionalities that do not include user interactions. The header of a ColumnFixture contains the name of

the function under test. The following row contains headers of data name columns. Input columns start from the left and are followed by output columns on the right.

The methodology to develop ESATs from the security robustness diagrams is based on the Model–View–Control (MVC) architecture posed by the security robustness diagrams. This means that the behavior of the use and misuse cases (usage patterns) is realized by a sequence of actions and hence the corresponding acceptance tests are driven by *action* fixtures. As actions are performed in the *action* fixtures, a timeline (state) of the usage pattern is progressing. At certain points the action flows (the timeline), other tests related to the checking of data lists (*row* fixtures) or calculations (*column* fixtures) are required. *Row* and *column* fixtures are stateless in the sense that their execution does not advance the timeline of the usage pattern. *Row* and *column* fixtures are hence embedded within the various *action* fixtures that progress the timeline of the usage pattern. Given the underlying (MVC) architecture of the security robustness diagrams, actions in a usage pattern start from an interface object, followed by the execution of one or more control objects. Control objects in turn access one or more entity objects, before relaying the output back to one or more interface objects.

Using the running example of the FSC subsystem, an ESAT is created to realize the corresponding HLSAT shown in Table 3. For the usage scenario involving the “remove application” misuse case, a number of fixtures are

created as shown in Fig. 13. Table 7 provides a brief description of the type and purpose of each fixture. As shown in Fig. 13, the acceptance tests begin with an *action* fixture which accesses the only given interface object *ApplicationViewer*. Interface objects are accessed to request a particular functionality to be relayed to a control object. It is common to provide the necessary parameters to the interface object also to be related to the control object to allow the control object to adequately perform its responsibilities. As shown in Fig. 14, the first *action* fixture “Enter”s various information before “Press”ing to display the active applications. The information entered should be displayed as entity objects in the security robustness diagrams. Meanwhile, the functionality to execute should be displayed as control objects in the security robustness diagrams. The function request is then relayed to control objects before “Check”ing the interface object once again that the applications are shown. At this point in the timeline of this usage scenario, it is necessary to a data list check; hence, a *row* fixture was utilized. The *row* fixture header labels the object to be evaluated. In this case, the data object to be checked is the *DisplayedApplications* object. The data object should also be present as an entity object in the security robustness diagram. It may be the case that a calculations type of check is required, and in this case, a *column* fixture will be utilized. The input and output data for the *column* fixtures should be present as entity objects in the security robustness diagrams.

ApplicationViewer		
Enter	Users.CurrentUser	Username
Enter	Users.CurrentUser	Password
Press	ApplicationViewer.DisplayActiveApplications()	
Check	ApplicationViewer.isDisplayActiveApplications()	True
ApplicationViewer.ApplicationList.DisplayedApplications		
Application-1		
Application-2		
Application-3		
ApplicationViewer		
Enter	ApplicationViewer.CurrentApplication	Application-1
Press	ApplicationViewer.Remove()	
ApplicationViewer.ApplicationList.DisplayedApplications		
Application-2		
Application-3		
ApplicationViewer		
Check	ApplicationViewer.CurrentApplication	Application-2
Press	ApplicationViewer.DisplayRemovedApplications()	
Check	ApplicationViewer.isDisplayRemovedApplications()	True
ApplicationViewer.RemovedApplicationList.DisplayedApplications		
Application-1		Username

Fig. 13 ESATs for security functionality against improper removal of an application



**Table 7** A description of fixtures used to test against the “Remove Application” misuse case

Fixture #	Type	Purpose
1	Action	The purpose of this first fixture is to retrieve the access credentials and to display the list of active applications
2	Row	This fixture assumes three applications are present in the system and checks for their existence
3	Action	The purpose of this fixture is to select an application and execute a remove operation upon it
4	Row	The purpose of this fixture is to check that the removed application is no longer active by checking for the other two remaining applications
5	Action	This fixture presents part of the mitigation behavior where the removed application is added to the “removed application list.” The fixture checks that the list is displayed
6	Row	This fixture presents the other part of the mitigation behavior by checking the contents of the “removed application list.” The fixture checks that the list contains the removed application as well as the user who executed the remove operation. The fixture assumes that there was no other previously removed application

Object names in a security robustness diagram are expected to remain as English words. This naming style is necessary to allow the modeler to focus on the modeling exercise whereby a solution is being delivered. It is ideally expected that the names of the objects will change to resemble candidate code-level naming. This evolution is necessary to produce machine-readable tests.

Upon completion of the development, the software system under test is that tested using the generated ESATs to verify that the system is performing correctly in case of misuse.

#### 4.4 Targeted coverage by the developed acceptance tests

The proposed approach aims to develop acceptance tests for different usage scenarios based on the usage pattern shown in Fig. 3. Therefore, the tests developed aims to achieve *activity path coverage* [32]. *Activity path coverage* is simply *Complete path coverage* but while observing Beizer’s loop “once” coverage strategy [33]. This means that the paths derived start from the initial node to the final node while covering each loop just once. As of the time of writing this paper, there has not been any empirical evaluation conducted that provides evidence of a better test coverage criteria for mis/use cases. Therefore, the *activity path coverage* was targeted as it was deemed the most appropriate given the scenario-based characteristic of mis/use cases.

#### 4.5 Utilization of the acceptance tests within a development methodology

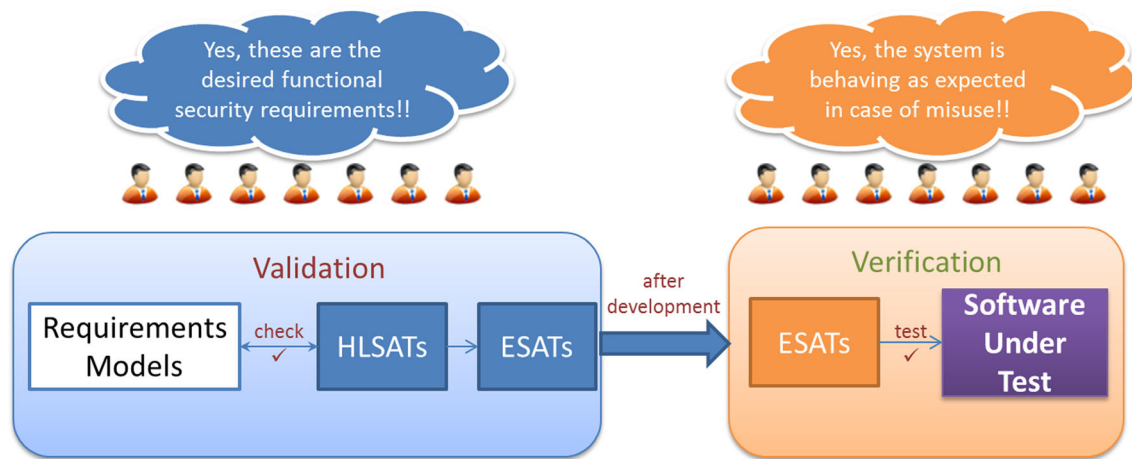
The proposed approach aims to improve the development methodologies of secure systems. The acceptance tests developed pre-development should be used as an effective medium to validate the desired functional security

requirements. Validation is achieved by eliciting feedback from the system stakeholders, especially customers, domain experts and end users. The functional security requirements are validated by the stakeholders when they confirm that the behavior exhibited by the system according to the acceptance tests is indeed an accurate representation of their desired functional requirements. After a system is developed, the executable security acceptance tests can be used as a verification mechanism to ensure that the system does indeed behave as expected in case of misuse. Figure 14 presents a visual summary of how the produced acceptance tests should be used within a secure software engineering development methodology.

## 5 Tool support

The execution of the ESATs is perhaps the only task that mandates tool support. However, automation support for other tasks performed in the proposed technique will be highly beneficial as it reduces the dependency on the skill of its user and ensures the accuracy and speed of its application. This section presents the Automated Secure Acceptance Testing Framework (ASATF). ASATF is a collection of tools that collectively provide automation to many of the tasks required by the proposed technique.

As shown in Fig. 2, the two initial inputs to the process are a domain model and a misuse case model. These two models can be developed by anyone of many UML tools. Tool support is heavily provided in the first phase of the process where three different tools are used in sequence. The first tool used is an extension of the tool ARBIUM (Automated Risk-Based Inspection of Use case Models). ARBIUM was developed in previous work [21]. The original purpose of this tool is to detect antipatterns in use case models. In fact, it can detect any type of pattern in use case models. This tool was extended to support the misuse



**Fig. 14** Utilization of the acceptance tests

case modeling notation and be able to detect the usage patterns (Fig. 3). The extended tool was named Ex-ARBIUM.

The tool presented in [26] uses the concept of model transformation to convert mis/use case descriptions into mal-activity diagrams [22]. Mal-activity diagrams are similar to UML activity diagrams but with an extended notation to model mal-activities (the inverse of activities) and misusers (the inverse of actors). The mal-activity diagrams are then used as input to the activity path generator (APG). APG is a direct implementation to the activity path generator algorithm presented by [32]. The outcome of APG is a set of single flows (usage scenarios) pertaining to each usage pattern previously detected by the tool Ex-ARBIUM. Given the single flows, they are analyzed carefully to determine the necessary inputs and outputs to actuate each usage scenario and complete the creation of the HLSATs. This final task of phase 1 requires analysis of the textual information, which needs to be performed manually; however, it can be guided using Functional Grammar theory as mentioned previously in Sect. 4.1. A tool named Essential Interactions Parser (EIP) was implemented based on the theory of Functional Grammar. EIP parses the textual descriptions to detect essential actions that can be candidates for control objects and nouns that can be candidates for inputs and outputs.

Phase 2 is concerned with performing security robustness analysis and developing security robustness diagrams. The secure robustness diagrams notation is also supported by many UML tools. Since the security robustness diagrams are based on the usage scenarios, which are embedded in the mal-activity diagrams generated by the model transformation tool of [26], the mal-activity diagrams were used to automatically generate a security robustness diagram containing control objects only. The

control objects have the appropriate coloring. Note that in order to achieve the appropriate coloring for mitigation control objects (green), the model transformation tool by [26] was extended since the original notation of mal-activity diagrams by [22] did not support mitigation activities. At this point, the security robustness diagram contains only control objects and is missing boundary and entity objects. As mentioned in Sect. 4.2, the addition of the missing boundary and entity objects is guided by the HLSATs and the objects available in the domain model. The addition of the boundary and entity objects is a manual activity.

In phase 3, the tool MUCAT (Misuse Use Case Acceptance Tester) is used to allow modelers to write and associate acceptance tests to their corresponding use/misuse cases. MUCAT also provides testers with the ability to reuse their tests. MUCAT is also an extension of the tool UCAT (Use Case Acceptance Tester) which was developed in previous work [21]. MUCAT interfaces with the Fit/Fitnesse framework to execute the security acceptance tests once development is completed. MUCAT yields the results of the tests in HTML form. It should be noted that the tool does not automatically create ESATs. Creating ESATs is a manual process that is explained in detail in Sect. 4.3.

## 6 The faculty search committee subsystem case study

This section presents the faculty search committee (FSC) subsystem case study to demonstrate how the proposed approach can be used to produce security acceptance tests using a misuse case model and its robustness diagrams. The FSC subsystem is currently being developed by professional developers at the IT department of King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

The purpose of the system is to facilitate the processes of receiving and reviewing applications for faculty position openings. The FSC subsystem is part of a much larger system that automates many other internal processes used by faculty members. In total, the system includes seven subsystems. The approach proposed in this paper was applied and validated using all subsystems. Demonstration of the feasibility and application of the proposed approach on the entire system will be very space consuming. Therefore, validation of the proposed approach will be demonstrated via the FSC subsystem only. A brief description of the other six subsystems is shown in Table 8. The term “Entity” in Table 8 refers to any use case, misuse case, actor or misuser, while the term relationship refers to any relationship type such as the include, extend, generalize, mitigate and threatens relationships.

In general, the users of the FSC subsystem include any faculty member of various departments. However, the faculty search committee members are expected to be the most involved users with the system. The use cases of the system were derived by the development team and in consultation with the faculty search committee. The faculty search committee consists of three members (faculty members) in addition to a chairman. It should be noted that the chairman of the faculty search committee is the author of this paper. The development team would apply the proposed technique to develop security acceptance tests in collaboration with the committee members. The committee members, as experts in the problem domain, would use the security acceptance tests to validate the intended security behavior of the system.

The underlying workflow for the review process begins with the department secretary scanning and uploading hardcopies of applications onto the system. The system tags the application with an ID and sends notifications to the committee members. The three committee members then independently review the application and render their recommendation. Only after the three members submit their review, the system notifies the committee chairman to review the application and also study the input from the committee members. The chairman then renders his decision based on his independent review of the application and the input from the other members. The chairman’s decision is treated as the committee’s decision. The decision is then forwarded to the department chairman for perusal.

The system offers some features (use cases) to facilitate the underlying workflow. The most basic feature is the ability of its users to view and review applications. The system allows its users to change their rendered decision or review of an application. The system also allows its users to redirect an application to another faculty member if deemed necessary. Based on the initial set of use cases, misuse cases were accordingly identified and the use case model evolved into a misuse case model. Misuse cases include improper change of decision or a review, improper removal of an application and improper redirection of an application to another department. Consequently, a new set of use cases were added to the misuse case model to fend against misuse of the system. The final misuse case model of the system is shown in Fig. 15. The misuse case model for the FSC subsystem contains 2 actors, 1 misuser, 6 use cases and 3 misuse cases. A brief description of

**Table 8** A brief descriptions of the other six subsystems

Subsystem	Description	Size of misuse case diagram	Involved users
Graduate applications	Subsystem is used to collect applications for graduate studies. The subsystem helps automate the review process	10 entities + 17 relationships	Externals and internal students + faculty + administration
Student petitions	This subsystem is used mainly by students to submit petitions for various reasons	5 entities + 11 relationships	Internals students + faculty + administration
Teaching scheduling	Subsystem is used to collect faculty preferences with respect to teaching and automate the scheduling process while observing the University rules and regulations	10 entities + 16 relationships	Faculty
Faculty promotion	This subsystem is used by faculty members who are eligible to apply for promotion. The system collects the relevant material from the applicant and allows an anonymous committee to process the promotion application	8 entities + 13 relationships	Faculty + administration
Conference attendance	This subsystem is used by faculty members to apply for conference attendance. Conference attendance at the host University must be approved by the Deanship of Scientific Research	4 Entities + 10 relationships	Faculty + administration
Maintenance requests	This subsystem can be used by all University staff and students to request various types of maintenance work to be performed	12 Entities + 20 relationships	Students + faculty + administration staff + maintenance staff

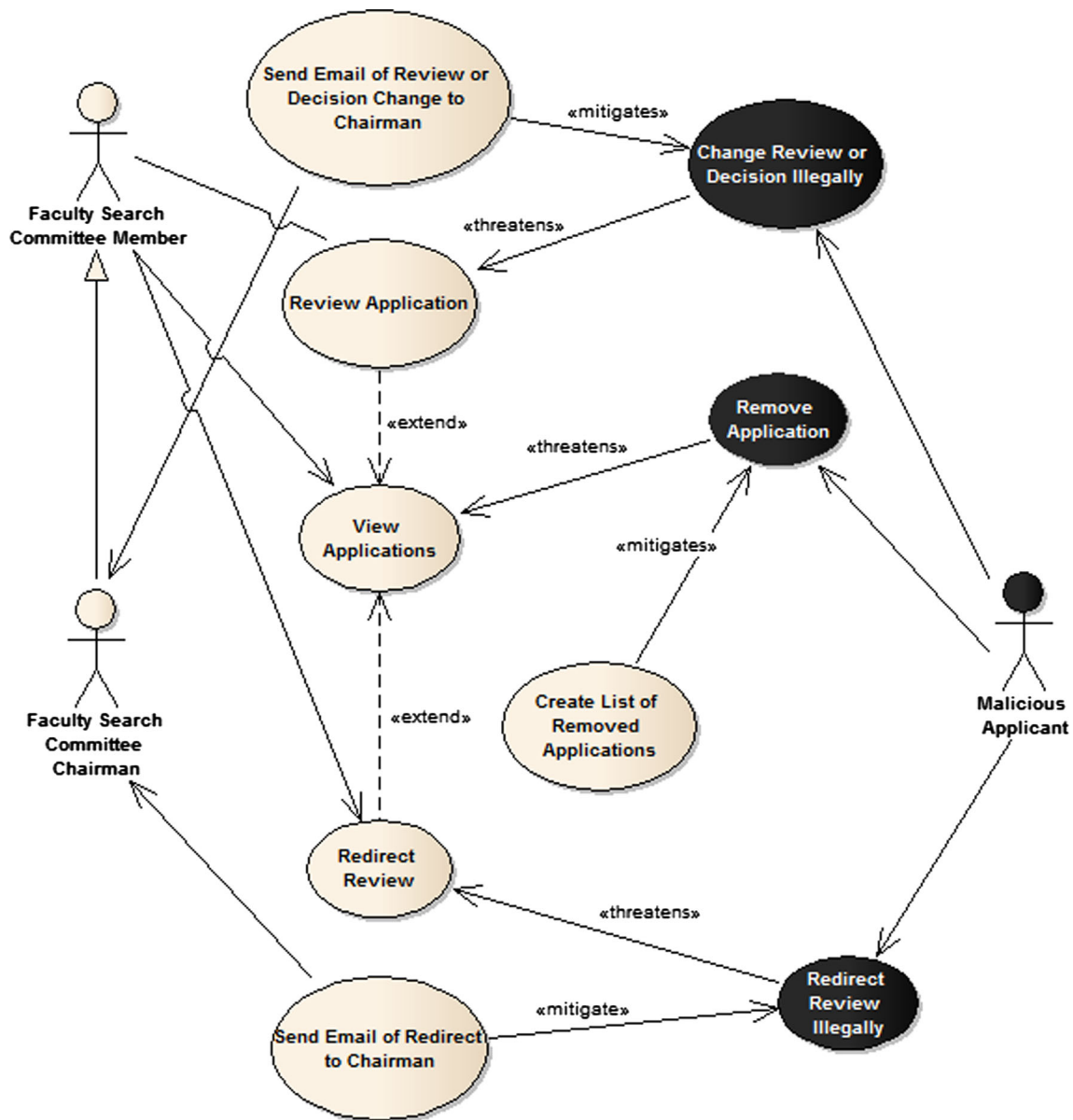


Fig. 15 Misuse case model of the FSC subsystem

each element in the model is provided in Table 9. The preliminary domain model of the FSC subsystem is presented in Fig. 16. The domain model is built through a brainstorming process. Table 10 provides a brief description of the classes contained in the preliminary domain model. Recall that the FSC subsystem is only a one component of the larger system that was developed to support faculty activities. As such, the misuse case model shown in Fig. 15 is only an excerpt of the much larger misuse case model that was built for the entire system. Similarly, the preliminary domain model developed for the entire system contains many more classes that those shown in Fig. 16.

The following subsections will demonstrate the application of the proposed technique to develop security acceptances tests based on the developed use/misuse cases. The aim of applying the proposed technique is to create a set of executable security acceptance tests that will cover misuse scenarios described by the three misuse cases. The following is an outline of the analysis performed to produce security acceptance testing for this case study:

- Section 6.1: The behavior of the “Change Review Decision Illegally” misuse case and its associated use cases is presented. This behavior is analyzed in the proceeding subsections.

**Table 9** Properties of the FSC subsystem misuse case model

Element	Purpose
<b>Actors</b>	
Faculty search committee member	A faculty search committee member is a licit user of the system. A faculty search member uses the system to mainly review applications and render decisions. A faculty search committee member may also redirect an application to another faculty member if deemed appropriate
Faculty search committee chairman	The faculty search committee chairman is a special type of faculty search committee member. He has additional privileges in comparison with regular faculty search committee members. The chairman can review changes in an application decision and requests for application redirects. The chairman has the power to cancel any changes if deemed appropriate
<b>Misusers</b>	
Malicious applicant	This is the sole misuser. A malicious applicant is a person who attempts to misuse the system to gain an unfair advantage for the application review process of a particular applicant. This can be done by redirecting application reviews, changing decisions and removing competing applications
<b>Use cases</b>	
View applications	This use case describes the necessary behavior to view all active applications
Review application	This use case is an extension to the “View Applications” use case. It describes the behavior of an application being reviewed by a faculty search committee member which culminates with a rendering of a decision
Redirect review	This use case is also an extension of the “View Applications” use case. It describes the behavior of an application being redirected by a faculty search committee member. Redirection requires the nomination of another faculty member to conduct the review
Send email of review or decision change to chairman	This is a mitigation use case whose purpose is to notify the chairman of any decision or review changes with respect to a particular application. This use case mitigates against the threat of illegally changing the decision or review of a particular application
Create list of removed applications	This is a mitigation use case whose purpose is to display the list of removed application. This use case mitigates against the threat of unfairly removing a competitive application
Send email of redirect to chairman	This is a mitigation use case whose purpose is to notify the chairman of an application review redirect. This use case mitigates against the threat of bias selection of an application referee
<b>Misuse cases</b>	
Change review or decision illegally	This misuse case describes the behavior initiated by a malicious application to illegally change the review or decision of a particular application
Remove application	This misuse case describes the behavior initiated by a malicious application to unfairly remove a competitive application
Redirect review illegally	This misuse case describes the behavior initiated by a malicious application to illegally redirect an application to a faculty member whom more likely to conduct a favorable review

- **(Phase 1)** A set of HLSATs are created to simulate the misuse behavior and test the mitigation behavior.
- **(Phase 2)** Security robustness analysis is performed and a corresponding security robustness diagram is created. Security robustness analysis identifies classes that correspond to the inputs and outputs stated in the HLSATs.
- **(Phase 3)** The identified classes are used to create ESATs to implement the HLSATs.
- Section 6.2 follows a similar structure to that of Sect. 6.1 to analyze misuse case “Redirect Review Illegally”.

#### 6.1 Change review or decision illegally: misuse case

This misuse case is initiated by the malicious applicant by providing a username that does not belong to them (unless the misuser is an insider). The system then provides the list of active application. The malicious applicant enters the name or ID of an application and the system retrieves it. The malicious applicant then changes decision or review previously rendered by a faculty search committee member. The expected mitigation behavior is then triggered by the system by emailing the committee chairman with details of the change. The chairman can then verify the change and either approve



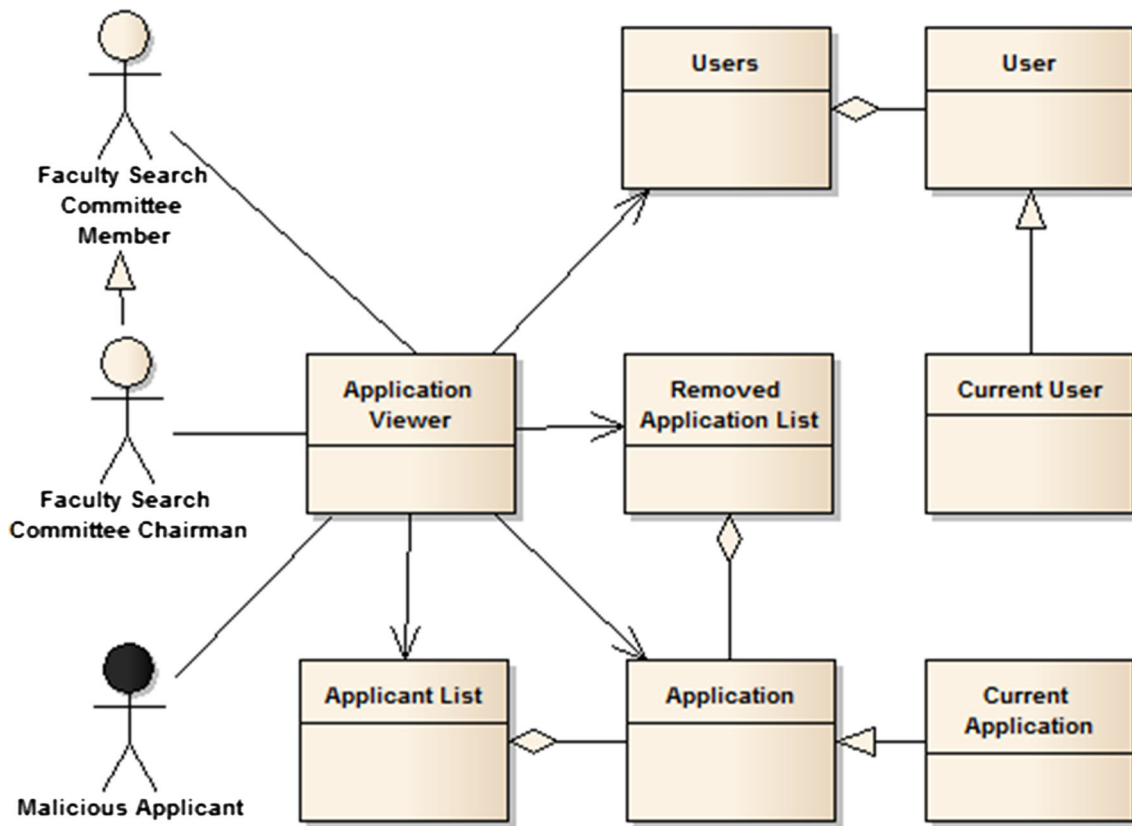


Fig. 16 The initial domain model of the system

Table 10 A brief explanation of the domain models classes

Class	Purpose
Application Viewer	This is the main interface to the system
Applicant List	Contains the list of all applications including removed applications
Removed Application List	Contains a list of applications that have been removed
Application	An application for employment by an applicant
Users	Registered user list containing faculty members
User	A particular registered user of the system
Current User	The current logged user in the system
Current Application	A current application retrieved from the application list for viewing

or disapprove it. The HLSAT shown in Table 11 was derived based on the behavior of the involved mis/use cases (Phase 1). The corresponding security robustness diagram developed is shown in Fig. 17 (Phase 2). Finally, the ESATs developed are shown in Fig. 18 (Phase 3). The ESATs shown below consists of four fixtures. Table 12 provides a brief description of the type and purpose of each fixture.

### 6.2 Redirect review illegally: misuse case

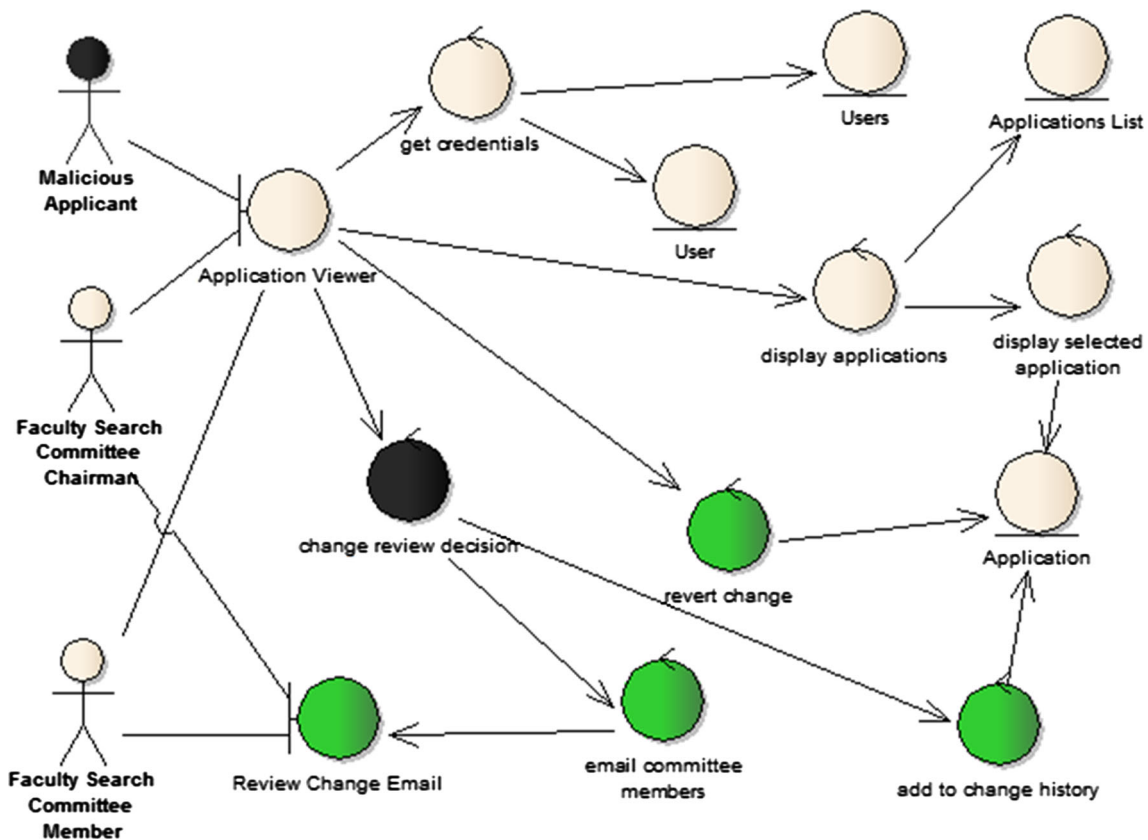
This misuse case is initiated by the malicious applicant by providing a username that does not belong to them (unless the misuser is an insider). The system then provides the list of active application. The malicious applicant enters the name or ID of an application and the system retrieves it. The malicious applicant provides the name of a new reviewer to review the application. The application is assigned a “redirected” status. The expected mitigation behavior is then triggered by the system by emailing the committee chairman with details of the redirection. The chairman can then verify the change and either approve or disapprove it. The HLSAT shown in Table 13 was derived based on the behavior of the involved mis/use cases (Phase 1). The corresponding security robustness diagram developed is shown in Fig. 19 (Phase 2). Finally, the EASTs developed are shown in Fig. 20 (Phase 3). The ESAT shown below consists of four fixtures. Table 14 provides a brief description of the type and purpose of each fixture.

### 6.3 Validation

Validation of the proposed is achieved on two fronts. Firstly, validation is achieved by determining how well the

**Table 11** The HLSAT for the “Change Review or Decision Illegally” misuse case

Test ID	Description	Expected results
Remove Application— <i>Basic Flow</i>	<i>Precondition:</i> Run FSC subsystem	<i>Output:</i> The selected application has its review decision changed
	<i>Input:</i> Username	<i>Output:</i> The selected application has its status changed
	<i>Input:</i> Password	<i>Output:</i> An e-mail sent to the committee members with the details of the change
	<i>Input:</i> ID of application to be changed	<i>Output:</i> The selected application has its status reverted
	<i>Input:</i> New status	<i>Postcondition:</i> Current Status = Old Status
	<i>Input:</i> Chairman username <i>Input:</i> Old status	



**Fig. 17** The security robustness diagram for the “Change Review or Decision Illegally” misuse case

tests created cover the behavior described in the use and misuse cases (as mentioned in Sect. 4.4). Secondly, validation is achieved by eliciting feedback from the stakeholders of the system, especially the customers whom most of them are also end users. The proposed approach is validated if the feedback received indicates that the outlined behavior of the system in terms of acceptance tests is indeed an accurate representation of their security requirements.

On the first front, validation is concerned with the coverage of the misuse behavior according to the usage patterns as presented previously in Fig. 3. Recall that the coverage strategy that was selected for our approach is the *activity path coverage* as mentioned previously in Sect. 4.4. For the FSC subsystem, the following usage patterns were covered (Table 15). As shown in Table 15, the usage patterns covered by the proposed approach encompass the entire misuse case model (business use cases, misuse cases

ApplicationViewer		
Enter	Users.CurrentUser	Username
Enter	Users.CurrentUser	Password
Press	ApplicationViewer.DisplayActiveApplications()	
Check	ApplicationViewer.isDisplayActiveApplications()	True
ApplicationViewer.ApplicationList.DisplayedApplications		
Application-1		
Application-2		
Application-3		
ApplicationViewer		
Enter	ApplicationViewer.CurrentApplication	
Press	ApplicationViewer.Review()	
Press	ApplicationViewer.Recommend()	
Check	Application.Status	Recommended
Check	Application.isDecisionChanged	False
Press	ApplicationViewer.Review()	
Press	ApplicationViewer.Reject()	
Check	Application.Status	Rejected
Check	Application.isDecisionChanged	True
Press	Application.addChangeToHistory()	
ApplicationViewer		
Enter	Users.CurrentUser	Committee Member-1
Enter	Users.CurrentUser	Committee Member-2
Enter	Users.CurrentUser	Committee Member-3
Enter	Users.CurrentUser	Committee Chairman
Press	ApplicationViewer.SendChangeDecisionEmail()	
Press	ApplicationViewer.RejectChange()	
Check	Application.Status	Recommended

**Fig. 18** ESATs for security functionality against improper overturn of a decision on an application

**Table 12** Descriptions of the fixtures used to test against the “Change Review or Decision Illegally” misuse case

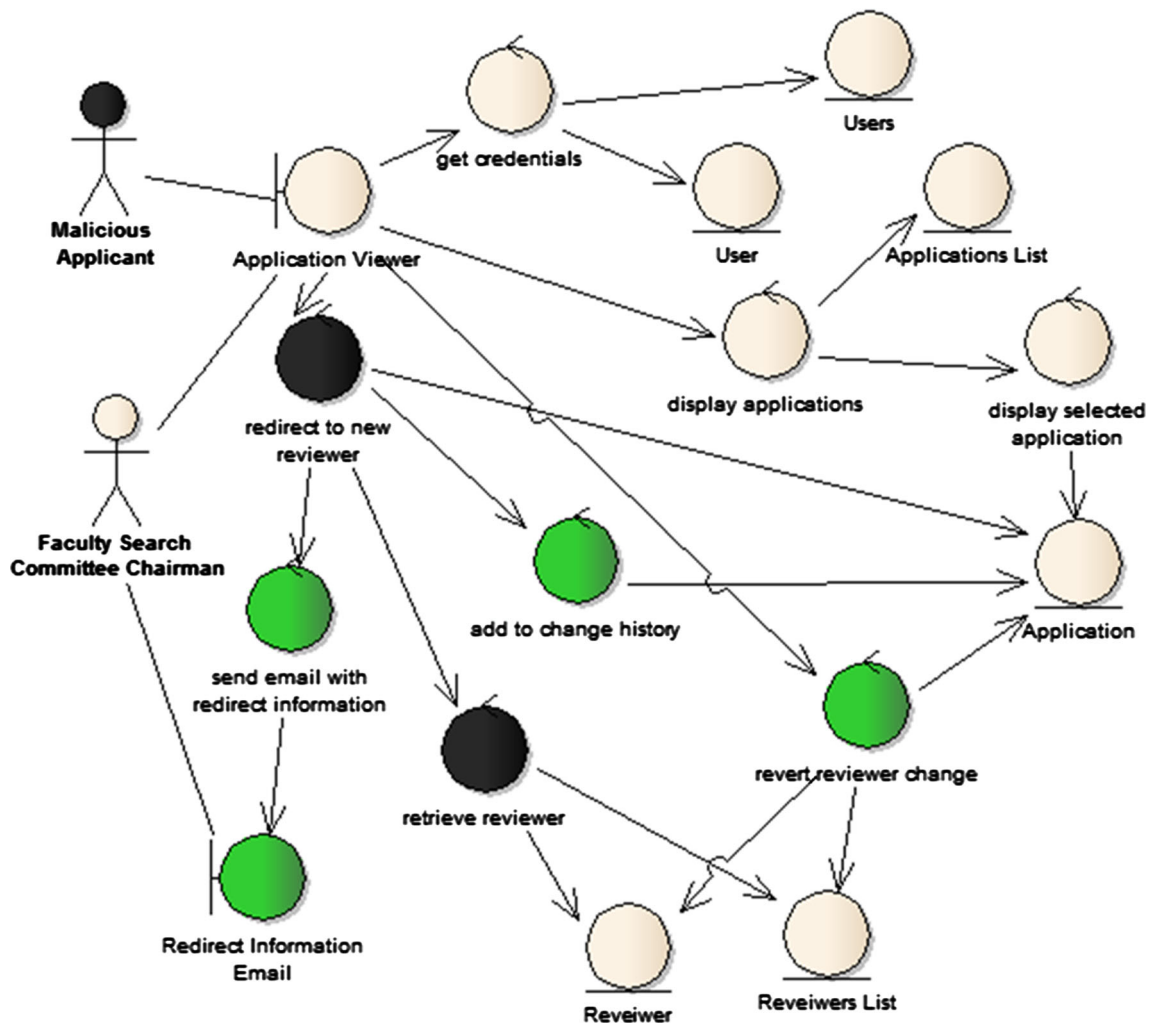
Fixture #	Type	Purpose
1	Action	The purpose of this first fixture is to retrieve the access credentials and to display the list of active applications. This fixture assumes three applications are present in the system
2	Row	This fixture assumes three applications are present in the system and checks for their existence
3	Action	This fixture begins with selecting an application for review. The application is then recommended. The fixture then executes another review where this time the decision has changed from a “recommend” to a “reject”
4	Action	This fixture presents the mitigation behavior. The four members of the committee and its chairman are notified with the decision change via e-mail. The fixture executes a change reject by the chairman and checks that the application’s decision has not changed

and mitigation use cases). The tests produced achieved 100 % activity path coverage for FSC subsystem. The tests produced for the remaining six subsystems also achieved 100 % coverage. The 100 % coverage is expected since the proposed approach is based on transforming the use/misuse case descriptions into mal-activity diagrams and then dissecting the mal-activity diagrams into single flows.

Upon applying the proposed approach to the entire system, a questionnaire was used to elicit feedback from the system stakeholders. The questionnaire was designed to evaluate the coverage of the tests produced for the usage patterns. The questionnaire also acts as a validation and verification mechanism of the security requirements. The questionnaire was given to all stakeholders of the system.

**Table 13** The HLSAT for the “Redirect Review Illegally” misuse case

Test ID	Description	Expected Results
Remove Application— <i>Basic Flow</i>	<i>Precondition:</i> Run FSC subsystem	<i>Output:</i> The selected application has its review decision changed
	<i>Input:</i> Username	<i>Output:</i> The selected application has its reviewer changed
	<i>Input:</i> Password	<i>Output:</i> An e-mail sent to the committee chairman with the details of the redirect
	<i>Input:</i> ID of application to be redirected	<i>Output:</i> The selected application has its reviewer change reverted
	<i>Input:</i> New status	<i>Postcondition:</i> Current Reviewer = Original Reviewer
	<i>Input:</i> Chairman username	
	<i>Input:</i> Original reviewer	



**Fig. 19** The security robustness diagram for the “Redirect Review Illegally” misuse case

This includes the potential end users as indicated in Table 8 and the development team. Questions Q1–Q6 shown below were designed to elicit data about the users’ perceived functional efficacy and correctness of the

proposed technique. The questionnaire contained closed-ended and open-ended questions. The closed-ended questions were designed to elicit quantitative data about the respondent’s perceptions of the tests for activity path

ApplicationViewer		
Enter	Users.CurrentUser	Username
Enter	Users.CurrentUser	Password
Press	ApplicationViewer.DisplayActiveApplications()	
Check	ApplicationViewer.isDisplayActiveApplications()	True
ApplicationViewer.ApplicationList.DisplayedApplications		
Application-1		
Application-2		
Application-3		
ApplicationViewer		
Enter	ApplicationViewer.CurrentApplication	
Enter	Users.NewReviewer	New Reviewer
Press	ApplicationViewer.Redirect()	
Check	Application.Status	Redirected
Check	Application.isDecisionChanged	False
Press	Application.addChangeToHistory()	
ApplicationViewer		
Enter	Users.CurrentUser	Committee Chairman
Press	ApplicationViewer.SendRedirectEmail()	
Press	ApplicationViewer.rejectChange()	
Check	Application.Reviewer	Original Reviewer

Fig. 20 ESATs for security functionality against improper redirect of an application

Table 14 A description of fixtures used to test against the “Redirect Review Illegally” misuse case

Fixture #	Type	Purpose
1	Action	The purpose of this first fixture is to retrieve the access credentials and to display the list of active applications. This fixture assumes three applications are present in the system
2	Row	This fixture assumes three applications are present in the system and checks for their existence
3	Action	Misuse behavior is simulated by this fixture by assigning a new review to the application before executing a redirect operation
4	Action	This fixture presents the mitigation behavior as the chairman committee is informed of the reviewer assignment change via e-mail. It simulates the chairman rejecting the change then checks that the current reviewer of the application remains as the original reviewer

Table 15 Usage patterns covered from the FSC subsystem misuse case model

Pattern	Original use case	Threatening misuse case	Mitigation use case
1	View applications	Remove application	Create list of removed applications
2	Review application	Change review or decision illegally	Send e-mail of review or decision change to chairman
3	Redirect review	Redirect review illegally	Send e-mail of redirect to chairman

coverage (Q1–Q3) as well as its clarity of describing the exact behavior of the system under various conditions (Q4–Q6). The open-ended questions were used to elicit qualitative data to validate and explain the various phenomena observed in the responses to the closed-ended questions. Questions Q1–Q6 are as follows:

**Q1** Did the tests fully cover the various usage scenarios of the system?

**Q2** Did the tests fully cover the various misuse scenarios of the system?

**Q3** Did the tests fully cover the various defensive countermeasures?

**Q4** Did the tests clearly explain how the system will be behaved when it is used?

**Q5** Did the tests clearly explain what happens in case of misuse?



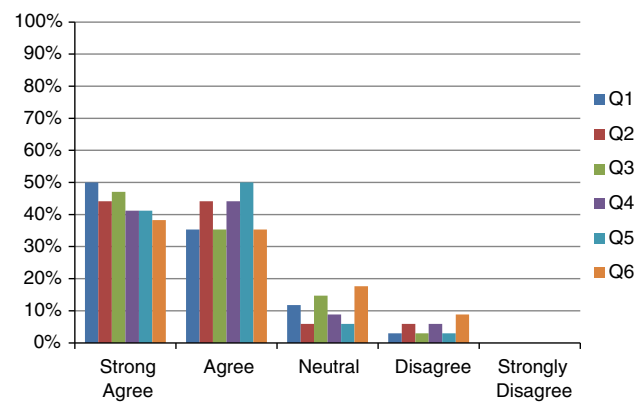
**Q6** Did the tests clearly explain how the system will be behaved in response to misuse?

Questions Q1–Q6 focus on the perceived functional correctness of the proposed technique by its users. These questions were created by the authors of the paper and not adapted from other questionnaires. The reason being that these questions are custom made to evaluate the specific functional goals of the proposed technique, which is not assumed to be readily available by other evaluation tools in the literature. The results shown are from correspondents whom were involved as participants in the development process for all seven subsystems. In total, we have received 34 complete responses shown in Fig. 21 based on a five-level Likert scale. The first author of this paper, whom was involved in the development project, did not partake in the questionnaire survey.

The results indicate that the majority of the stakeholders find that the behavior prescribed by the acceptance tests do indeed resemble the desired functional security requirements (Q1–Q3). The results also indicate that the majority of the stakeholders also find that the tests provide a clear explanation as to how the system will exactly behave in each usage scenario (Q4–Q6). However, there were some respondents whom expressed that the tests were inadequate or they were undecided about their efficacy. We examined the responses to the qualitative questions to shed further light into this issue. The responses to the qualitative questions reveal that the uncertainty felt by the respondents were mainly due the fact that acceptance tests created did not include more examples of stringier validation of the functional security requirements. This, however, does not negate the fact that current test suites provide full activity path coverage.

The questionnaire also contained questions to elicit data regarding cognitive dimensions of the proposed technique. The cognitive dimensions study will provide additional insight into the perceived usability of the proposed technique by its users. The questions have been adapted from [34]. The cognitive dimensions evaluated are described below:

- *Visibility* Is it easy to see the behavior representing the use and misuse cases?
- *Viscosity* Is it easy to make changes?
- *Diffuseness* The notation is precise and space-economic
- *Hard mental effort* Does it require hard mental effort to do?
- *Error proneness* It is easy to make mistakes or errors
- *Hidden dependencies* Are dependencies visible?
- *Progressive evaluation* Is it easy to stop and check my work so far?
- *Premature commitment* Is it easy to work in any order I like when using the approach?



**Fig. 21** The results to questions Q1–Q6

**Table 16** The results of the cognitive dimensions questions

Cognitive dimension	Strongly agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly disagree (%)
Visibility	91.18	8.82	0.00	0.00	0.00
Viscosity	88.24	8.82	2.94	0.00	0.00
Diffuseness	97.06	2.94	0.00	0.00	0.00
Hard mental effort	5.88	14.71	14.71	55.88	8.82
Error Proneness	11.76	8.82	26.47	44.12	8.82
Hidden dependencies	23.53	58.82	8.82	8.82	0.00
Progressive evaluation	47.06	47.06	5.88	0.00	0.00
Premature commitment	0.00	0.00	0.00	2.94	97.06

The results of the cognitive dimensions are shown in Table 16 based on a five-level Likert scale. The questionnaire also contained open-ended qualitative questions to allow us to explore in more detail the results shown in Fig. 21 and Table 16. The *visibility*, *viscosity* and *diffuseness* questions scored highly in favor of the proposed technique, which explains the favorable evaluation results it received with respect to its functional effectiveness (as shown in Fig. 21 for Q1–Q6). The *hard mental effort* question received lesser scores as the participants have indicated that applying the technique requires some degree of skill on part of its users, in particular, developing the necessary models and progressing from one phase to the next. The result for the *error proneness* question is in line with the *hard mental effort* question. Naturally, if a technique requires some degree of skill to apply effectively, then some degree of error proneness will exist. For the *hidden dependencies* question, the participants indicated that a traceability framework should be in place to better link artifacts to their sources. The result for the *progressive*

*evaluation* question clearly indicates that the users of the technique had no trouble monitoring their work. Finally, the result for the *premature commitment* question was rather expected. The proposed technique mandates a certain order of which tasks should be performed. This leaves users with room to change the order of their work only within tasks and not between tasks.

#### 6.4 Threats to validity

This section presents the threats to the validity of the study in accordance with the standard classification [35].

##### 6.4.1 Conclusion validity

A main conclusion threat to validity in this case study is that it represents only one sample, not enough to draw a strong empirically proven conclusion. While the case study deals with many personnel and many subsystems, a proper empirical evaluation would be necessary to provide the ultimate confidence that the treatment applied (the proposed approach) is responsible for the outcome (the success of the project). This lack of empirical evidence leads in turn to a number of external threats to validity (discussed in Sect. 6.4.4). We have listed a number of useful empirical evaluations that can better validate certain aspects that cannot be validated by the case study. The empirical evaluations we suggested are discussed in the last paragraph of Sect. 7.

Another conclusion threat to validity relates to the five-scale Likert scale used in the usability study. There is always a possibility that the participants did not score the different questions accurately due to misunderstanding of the questions or misunderstanding of the criteria to choose a particular answer. To mitigate against these threats, the participants were given an explanation of the various questions as well as an explanation to the various criteria.

##### 6.4.2 Internal validity

The forefront internal threat to validity is the involvement of one of the authors of this paper in the development project. Although the author did not partake in the questionnaire component of the evaluation, the author's involvement, given his familiarity with the technique must have surely improved the correctness of its application. However, the author was only involved in the development of one of the seven subsystems, the FSC subsystem. The author only provided some training to the development team whom applied the technique to the remaining six subsystems without any direct supervision from the author. The evaluation results shown in this case study are for the entire set of seven subsystems, and therefore, the author's

involvement in the development of one subsystem is not believed to have significantly skewed the results of the evaluation.

Another threat to validity is the quality level of the misuse case models used. The proposed technique requires a misuse case model to be developed, but it does not guide the development of high-quality misuse case models. The misuse case models used in the case were of high quality. However, if the misuse case models used are low quality, it is expected that the success of the technique will be significantly reduced.

##### 6.4.3 Construct validity

As indicated by the usability evaluation results, the success of the proposed technique is highly dependent on the skill of the person applying it. This is, however, the case for almost all requirements engineering approaches, and the proposed approach is no exception. We firmly believe, however, that with sufficient training and without a steep learning curve, requirements engineers will be able to apply the technique effectively.

Another construct validity stems from the fact that the software engineers applying the proposed approach are professionals. It is common the professionals divert from the prescribed instructions and resort to their prior experience. To mitigate this threat, the authors have closely reviewed their outcomes to ensure that they have following the prescribed instructions.

Although the participants of the case study were not informed that the authors of this paper are the inventors of the methodology used, it is safe to assume that the participants had inferred this fact on their own. This may lead to bias and influence the results obtained in the user study. In order to mitigate this threat, the participants were informed (verbally) that there is no right or wrong answers and that they should only be answering the questions based on their true perceptions.

##### 6.4.4 External validity

It is often the case that the size of systems considered in research case study is smaller than artifacts used in full-scale industrial settings which reduces the confidence in generalizing the results to industry. We believe our methodology can scale-up to larger and more complex systems. However, an empirical evaluation is certainly needed to prove this conjuncture.

Another threat to validity posed by this case study is the familiarity of its domain to the general public. The academic domain is well known, and its inner workings cannot be considered to be of extreme sophistication. It is expected that the success of the technique will be reduced

if it is applied to an unfamiliar domain or by a person who is unfamiliar with the given domain. The skill level of the participants in this case study is an inherent external threat to validity. This is due to the fact that it is not safe to assume that all other development groups in industry will have the same skill level as the participants of this case study. Other development groups may have higher or lower skill levels, which in turn will affect the effectiveness of the application the proposed approach.

The misuse case models used in this case study were created in cooperation with one of the authors of this paper who has a decade of use case and misuse case modeling research expertise. Hence, the misuse case models used as input to the methodology are of high quality. Misuse case models used in industrial settings are likely not to be of the high standard used in this case study. Following the garbage-in–garbage-out rule of computer science, the success level observed in this case study may not be observed in other projects.

## 7 Conclusion and future work

In the paper, we presented an approach to develop acceptance tests for early-stage validation of functional security requirements. The proposed approach makes use of artifacts that are developed at an early stage in the development life cycle, namely the domain model and misuse case model. The approach applies the security robustness analysis technique on the given domain model and misuse case models. The developed acceptance tests are executable and reusable. The acceptance tests are produced in a systematic manner to be much more comprehensive in comparison with an ad hoc approach to acceptance tests creation. The feasibility of the proposed approach was demonstrated using real-world system—the FSC subsystem used by various departments at King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

Tests are first created at high level to serve a dual purpose. Firstly, HLSATs are the basis for creating ESATs. Secondly, HLSATs provide a technically simple means for early validation of the security requirements, which encourages more involvement by customers. HLSATs can be developed at the early stages of the development life cycle without the need to wait for object-level information to become available.

Misuse case models adhere to a relatively small set of syntax rules. The core component of misuse case models is their unstructured textual descriptions. It is naturally very difficult to devise fully automated approaches that require the analysis of unstructured natural language. Therefore, human judgment is required during the application of the proposed approach. In particular, human judgment is

required to analyze the textual descriptions in order to decompose the stated behavior into *steps*, which is a requisite step to determine the usage scenarios, needed inputs and expected outputs. Naturally, processes that require human judgment are subjective. The approach is also dependent on the quality of the given misuse case and domain models. Therefore, the success of the proposed approach is dependent on the skill level of the analysts applying it and the quality of the given models. The issue with the reliance on human judgment can be remedied by prescribing the use of structured misuse case descriptions, such as the SMCD (Structured Misuse Case Descriptions) structure developed in earlier work [36]. SMCD structured misuse case descriptions will greatly reduce the subjectivity issue. Moreover, the use of the SMCD, which was specifically designed to reduce inconsistencies in misuse case descriptions, will ensure a certain level of quality better than that in unstructured descriptions. Therefore, future work can be directed toward modifying and improving the current approach by set it to leverage the SMCD structure.

A number of empirical studies can be performed to investigate possible generalization of the results that could not be supported from the current case study. For example, an empirical study can be performed using participants with varying experience and skill levels. Another empirical study can be performed using misuse case models of varying quality levels. It will also be interesting to perform other case studies where the authors are not involved in order to eliminate any potential for bias. Finally, an empirical evaluation can be useful to investigate the effect of using misuse case models representing uncommon domains.

**Acknowledgments** The authors would like to acknowledge the support provided by the Deanship of Scientific Research (DSR) at King Fahd University of Petroleum and Minerals (KFUPM) for funding this work through project No. IN111028.

## References

1. Jürjens J, Juerjens J (2005) Secure systems development with UML. Springer, Berlin
2. Sauv e JP, Abath Neto OL, Cirne W (2006) Easyaccept: a tool to easily create, run and drive development with automated acceptance tests. In: Proceedings on 2006 international workshop automation and software testing ACM, pp 111–117
3. Mantei MM, Teorey TJ (1988) Cost/benefit analysis for incorporating human factors in the software lifecycle. Commun ACM 31:428–439
4. Cohn M (2004) User stories applied: for agile software development. Addison-Wesley Professional, Reading
5. Sindre G, Opdahl AL (2005) Eliciting security requirements with misuse cases. Requir Eng 10:34–44
6. Alexander I (2002) Initial industrial experience of misuse cases in trade-off analysis. In: Requirement Engineering 2002—proceedings on IEEE joint international conference, pp 61–68

7. Den Braber F, Dimitrakos T, Gran BA et al (2002) Model-based risk management using UML and UP. *Issues Trends Inf Technol Manag Contemp Organ*
8. Houmb SH, Den Braber F, Lund MS, Stølen K (2002) Towards a UML profile for model-based risk assessment. In: *Critical system development with UML—proceedings UML'02 workshop*. Citeseer, pp 79–91
9. Karpati P, Redda Y, Opdahl AL, Sindre G (2014) Comparing attack trees and misuse cases in an industrial setting. *Inf Softw Technol* 56:294–308. doi:10.1016/j.infsof.2013.10.004
10. Rasputnig C, Opdahl A (2013) Comparing risk identification techniques for safety and security requirements. *J Syst Softw* 86:1124–1151
11. Sindre G, Opdahl AL, Brevik GF (2002) Generalization/specialization as a structuring mechanism for misuse cases. *Proc. 2nd symposium on requirements engineering: information security SREIS'02*, Raleigh, North Carol
12. Sindre G, Opdahl AL (2001) Templates for misuse case description. In: *Proceedings of 7th international workshop on requirements engineering: foundation for software quality*. REFSQ2001 Switz
13. Kroll P, Kruchten P (2003) *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional, Reading
14. Kulak D, Guiney E (2000) *Use cases: requirements in context*. Addison-Wesley, Reading
15. Basanieri F, Bertolino A, Marchetti E (2002) The cow\_suite approach to planning and deriving test suites in UML projects. *« UML » 2002—unified modeling language*. Springer, Berlin, pp 383–397
16. Briand L, Labiche Y (2002) A UML-based approach to system testing. *Softw Syst Model* 1:10–42
17. Nebut C, Fleurey F, Le Traon Y, Jezequel J-M (2006) Automatic test generation: a use case driven approach. *Softw Eng IEEE Trans* 32:140–155
18. Ryser J, Glinz M (1999) A scenario-based approach to validating and testing software systems using statecharts. In: *Proceedings 12th international conference on software, systems engineering and their application*.
19. International Institute of Business Analysts: Business Analysts Body of Knowledge. [www.iiba.org/babok-guide.aspx](http://www.iiba.org/babok-guide.aspx). Version 2.0. Last accessed March 2014
20. El-Attar M, Elish MO, Mahmood S, Miller J (2012) Is in-depth object-oriented knowledge necessary to develop quality robustness diagrams? *J. Softw* 7(11):2538–2552
21. El-Attar M, Miller J (2010) Developing comprehensive acceptance tests from use cases and robustness diagrams. *Requir Eng* 15:285–306
22. Sindre G (2007) Mal-activity diagrams for capturing attacks on business processes. *Requirements engineering: foundation for software quality*. Springer, Berlin, pp 355–366
23. Kariyuki, S. et al (2011) Acceptance testing based on relationships among use cases. In: *Proceedings of 5th world congress for software quality*, 2011.
24. Stephens M, Rosenberg D (2010) *Design Driven Testing: Test Smarter, Not Harder*. Apress
25. Roubtsov S (2006) Use case-based acceptance testing of a large industrial system: approach and experience report. In: *Proceedings of testing: academic and industrial conference—practice and research techniques*, 2006
26. El-Attar M (2014) From misuse cases to mal-activity diagrams: bridging the gap between functional security analysis and design. *Softw Syst Model* 13:173–190. doi:10.1007/s10270-012-0240-5
27. Dik SC (1997) *The theory of functional grammar: the structure of the clause*. Walter de Gruyter
28. El-Attar M (2010) Developing precise misuse cases with security robustness analysis. *SEKE*. pp 571–576
29. Rosenberg D, Scott K (1999) *Use case driven object modeling with UML*. Springer, Berlin
30. Mugridge R, Cunningham W (2005) *Fit for developing software: framework for integrated tests*. Pearson Education
31. Selenium Browser Automation: Selenium IDE. <http://docs.seleniumhq.org/>. Version 2.5.0. Last Accessed March 2014
32. Kundu D, Samanta D (2009) A novel approach to generate test cases from UML activity diagrams. *J Object Technol* 8:65–83
33. Beizer B, Wiley J (1996) *Black box testing: techniques for functional testing of software and systems*. IEEE Softw 13:98
34. Kutar M, Britton C, Wilson J (2000) Cognitive dimensions an experience report. *Proceedings of the twelfth annual meeting of the Psychology of Programming Interest Group, Memoria, Cozenza Italy 2000*:81–98
35. Wohlin C et al (2000) *Experimentation in software engineering—an introduction*. Kluwer, Dordrecht
36. El-Attar M (2012) Towards developing consistent misuse case models. *J Syst Softw* 85:323–339

Copyright of Requirements Engineering is the property of Springer Science & Business Media B.V. and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.