USING SECURITY ROBUSTNESS ANALYSIS FOR EARLY STAGE VALIDATION OF FUNCTIONAL SECURITY REQUIREMENTS

BY

AKRAM ABDUL-GHANIL HEZAM

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

DECEMBER, 2013

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **AKRAM ABDUL-GHANI HEZAM** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN MECHANICAL ENGINEERING.**

[illegible]

Dr. Mohamed El-Attar
(Advisor)

[illegible]

Dr. Adel Ahmed
Department Chairman

[illegible]

Dr. Mahmoud Elish
(Member)

[illegible]

Dr. Salam A. Zummo
Dean of Graduate Studies

[illegible]

Dr. Moataz Ahmed
(Member)

6/1/14

Date

DEANSHIP OF GRADUATE STUDIES عمادة الدراسات العليا

Dedication

To my beloved parents, wife, brothers, sisters andmy children.

ACKNOWLEDGMENTS

Thanks and all praise be to Allah (azzawagal)who created me, blessed me with health and who grant me success.

I would like to state my deep appreciation and gratitude to my thesis advisor Dr. Mohamed El-Attarwho has always beenproviding kindestsupport, continual encouragement and supervision.

I also would like to acknowledge my committee members: Dr. Mahmoud Elishand Dr. Moataz Ahmed fortheir valuable advices and guidance.

I also acknowledge KFUPM and Information and Computer Science (ICS) as well as the all faculties who taught and helped me.

Finally, I acknowledge Kingdom of Saudi Arabia for providing me MS scholarship and for all different facilitations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

OO	:	Object-Oriented
IDS	:	Intrusion Detection System
XP	:	Extreme Programming
AAT	:	Automated Acceptance Test
Fit	:	Framework for Integrated Test
HTML	:	HyperText Markup Language
BAs	:	business analysts
OCL	:	Object Constraint Language
BABOK	:	Business Analysts Body of Knowledge
SUT	:	System Under Test
UML	:	Unified Modeling Language
HLSATs	:	High-Level Security Acceptance Tests
ESATs	**:**	Executable Security Acceptance Tests
ICS	:	Information and Computer Science
IT	:	Information Technology
SMCD	:	Structured Misuse Case Description

UC	:	Use Case

ABSTRACT

Full Name	:	Akram Abdul-GhaniHezam
Thesis Title	:	Using Security Robustness Analysis for Early Stage Validation of Functional Security Requirements
Major Field	:	Information and Computer Science
Date of Degree	:	December, 2013

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. Moreover, acceptance testing is only leveraged in agile development methodologies. This thesis 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. It also requires a skill set that can be effectively used and communicated by non-technical stakeholders. The feasibility of the proposed approach is demonstrated by applying it to a real-world system – the Faculty Search Committee 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.

ملخص الرسالة

الاسم الكامل:أكرم عبد الغني حزام محمد

عنوان الرسالة:استخدام تحليل متانة الحماية للتحقق من صحة متطلبات حماية أنظمة المعلومات في مراحلها المبكر

التخصص:علوم الحاسب الآلي

تاريخ الدرجة العلمية:ديسمبر 2013

الأمن في الوقت الحاضر هو شرط لا غنى عنه في أنظمة البرمجيات. تركز عمليات هندسة البرمجيات التقليدية في المقام الأول على متطلبات العمل، وتترك الأمن إلى مرحلة لاحقة حيث يتم معالجتها عبر آليات دفاعية عامة. وتعتبر هذا الطريقة غير كافية، وتحتاج متطلبات حماية أنظمة البرمجيات لهندستها بطريقة مماثلة لهندسة المتطلبات الوظيفية العادية. حيث تتطلب متطلبات الأمن إلى أن تستخرج وتحلل وتحدد ويتحقق من صحتها في المراحل المبكرة من عملية تطور البرمجيات. إذا لم يتم التحقق من صحة المتطلبات لحماية أنظمة المعلومات بشكل صحيح، فسوف نواجه خطر تطوير نظام غير آمن، وبالتالي سيكون غير صالح للاستعمال .اختبار قبول البرمجيات يعتبر إسلوب فعال للتحقق من صحة المتطلبات. ومع ذلك، فإن هذا المنهج المخصص لتطوير اختبارات القبول يعاني من تجاهل بعض الاختبارات الهامة. وعلاوة على ذلك، يتم استخدام اختبار القبول فقط في أطر عمل أجايل لتطوير البرمجيات على وجه العموم. وتعرض هذه الرسالة طريقة منهجية لتطوير اختبارات القبول القابلة للتنفيذ بحيث تكون موجهة خصيصا لعمليات هندسة البرمجيات الآمنة المستندة النماذج. وتقوم هذه الطريقة على استخدام الوثائق الناتجة من المراحل المبكرة لتطوير البرمجيات ، وهي حالة سوء الاستخدام ونماذج المجال، والرسوم التصميمية. كما يتطلب مجموعة من المهارات التي يمكن استخدامها بشكل فعال وإيصالها من قبل المستفيدين من النظام من غير التقنيين. وتتجلى جدوى هذه الطريقة المقترحة عن طريق تطبيقها على نظام في الواقع – نظام البحث عن أعضاء هيئة تدريس في كلية. وقد أظهرت النتائج أن مجموعة شاملة من اختبارات قبول حماية النظام يمكن تطويرها استنادا إلى نماذج حالة إساءة الاستخدام للتحقق من صحة متطلبات حماية أنظمة المعلومات في مراحلها المبكرة.

CHAPTER 1

INTRODUCTION

1.1 Introductory Background

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 towards 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 is insufficient and will likely cause costly reworks in addition to any intangible consequences caused by a security breach [Jürjens, Matulevicius,Røstad]. 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 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[Sauvé]. 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 [Agarwal, Good, Goodhue, Gould, Gould-b, Gould-c, Mantei, Rosson].

The current state of practice finds acceptance testing almost exclusively deployed in agile processes. Since acceptance tests are developed based on requirements artifacts, therefore they are often constructed from user stories [Cohn]. 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 Sindreand Opdahl [Sindre] to account for functional security requirements. The extended modeling technique is named misuse case modeling [Sindre]. The misuse case modeling technique has emerged during the past decade as a promising technique to elicit and model functional security requirements [Alexander, den Barber, Houmb, Sindre]. Misuse cases define improper usage scenarios of a system by its external entities that may lead to harmful consequences in a very similar way by in which use cases describe legitimate usage instances. The literature has already reported a number of successful industrial experiences with misuse case modeling [Alexander, Alexander-b, Ekremsvi, Mæhre, Diallo, Stålhane]. 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 thesis 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.

1.2 A Brief Introduction to Misuse Case Modeling

The misuse case modeling technique was introduced by Sindre and Opdahl in 2000 [Sindre] 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 formally defined in [Sindre] 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"*. [Sindre]
>
> Misuser: *"An actor that initiates misuse cases, either intentionally or inadvertently"*. [Sindre]

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 [Sindre].

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 [Sindre-b, Sindre-c]. The proposed templates are often based on popular use case templates such as in [Cockburn, Kroll, Kulak]. 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. Figure1 shows an example of a misuse case diagram.

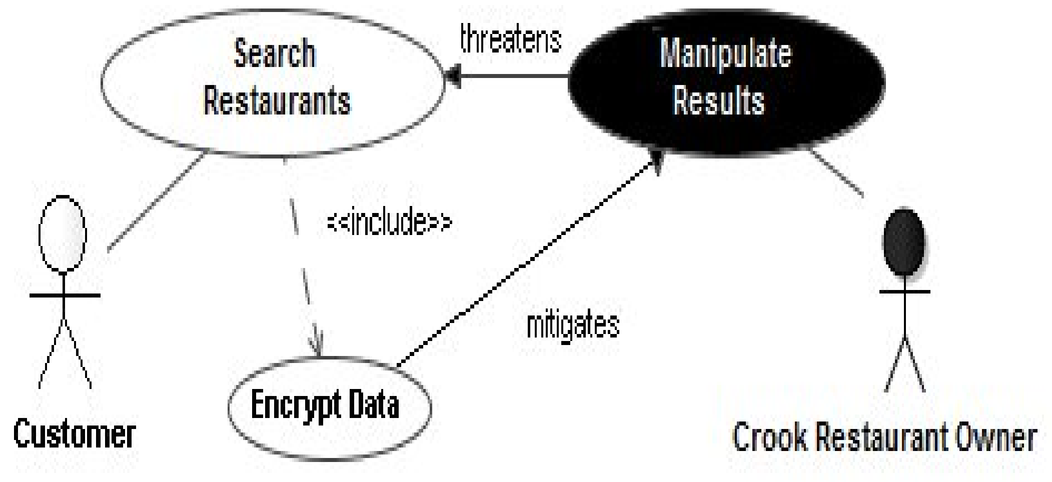


Figure 1A misuse case diagram example

The misuse case diagram shown in Figure 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.

1.3 A Brief Introduction to Use Case Modeling

Usecases wereintroducedby Jacobson (Jacobsonet al., 1992) [Jacobson].A use case describes the communications between outside actors and the system in order to realize the user’s goal [Sindre-c]. “*An actor might be a person, a company or organization, a computer program, or computer system hardware, software, or both.*”[Cockburn]. As we know, use cases can be used as famous techniquefor communicating, eliciting, and documentingrequirements. Use cases can be used to express functional requirements of the system, but theygive less expression for dealing with extra functional requirements, such as security requirements [Cockburn, Sindre-c, Rosenberg, Schneider, Holopainen,Mattingly]. Use cases, by their nature, focus on what the system should provide, and have less to describe what the system should not provide.

A use case in general illustrates functions of the system that should be able to execute. There are two types ofrelationship between use casesincludeand extend relationship. Include relationship is a type of relation between two use cases such that the base use case will contain the functionality for another use case.Extend relationship is a type of relation between two use cases such that use case will extendthe behavior of base use case[Cockburn,Sindre-c, Rosenberg,Rumbaugh].

1.4 A Brief Introduction to Acceptance Tests

"*Developers write unit tests to determine if their code is doing things right. Customers write acceptance tests to determine if the system is doing the right things*."[Miller]

Acceptance test term was introduced by Kent Beck in his book test-driven development by example [Beck]. Acceptance testing can be considered as a type of test, conducted to check if the requirements specification of the system met the customer's expectations. Acceptance testing evaluates the system under test with end user requirements [El-Attar, Briand, Beck]. The team development can use acceptance testing as a validation process to ensure that the functions of the system under test as planned; while system testing can be used as a verification process to guarantee that they will be creating the right system that will satisfy the end user requirements. Acceptance test is similar to system test in case of they are testing the whole system as the black box. System test and acceptance test do not need to know the underlying code of the system. However, system testing and acceptance test remain separate processes [El-Attar, Beck].

Testing in agile software development [El-Attar] is primarily key as it allows reflectivity andImproves communication between developers. Acceptance tests express the customer's needs. The acceptance tests can be used to give the customer self-assurance that the application has the essential features and that they perform correctly. In general we consider the project is ready or done when it passes all the acceptance tests.Acceptance tests can be considered as a "contract" between the developers and the customer. Preserving those tests, executing themregularly, and manipulating them as requirements modification,proves that there has been no break of contract [Rogers].

The benefits of a software development team from acceptance tests as the followings:

1. Acceptance testsare used to capture user requirements in a rightsupportable way, and they measure how well thesystem meets those requirements.
2. They uncover problems that unit tests omission.
3. They offer an off-the-shelf description of how "done"the system is.

As we know that recognizing user requirements is serious to the achievement of a project. If your system doesn't ensure whatusers need, it might be officially stylish but basically worthless. The difficult is in supposing, as severalapproaches do, that in-depth specifications will benefit.

There is solid evidence for the clue that extensive requirements specifications are unmanageable. Even if exhaustivespecifications were possible, they do not assurance that your system will do what users need [Miller,Mugridge].

Acceptance tests focus on both issues. First, acceptance tests can develop as the system develops, capturing user requirements as they evolve. Second, acceptance tests can be authorized directly – if a test passes, the system achieves the user requirements [Rogers].

1.4.1 Challenges in acceptance testing:

Acceptance testing looks very simple, but it can be a difficult to do it correct. The main topics to report are who writes tests, when they write tests, when tests are execute, and how to track tests.

Who writes the tests?

The business side of the project must be responsible ofwriting tests, or collaborate with the progress team to write them. The “business side” of the project might be the customer, other fellows of the customer’s association (such as QApersonnel, business analysts, etc.), or mixture of the two. The extreme programming(XP) customer eventually is in charge of creating the tests are written, nevertheless of who writes them. Business people must be able to write test in a language that they know. This language can be used to describe things that make the client or customer satisfied[Crispin,Miller,Mugridge].

When to write the tests

Business people must write acceptance tests before designers have completely implemented code for the properties being tested. Usingthese requirements in this directly will help to reduce miscommunication between the client and the development team. It also helps save the design easy, greatly as writing unit tests before writing source code.

The development team must transcribe just sufficient code to acquire the property to pass. It is essential for business people to transcribe tests before the "final product" is done, but they must not write them too primarily [Crispin, Miller].

When to run the tests

Tests must be able to execute mechanically at a configurable regularity, and by hand as desired. Once the tests havebeen written, the team mustexecute them regularly. This must develop as much as portion of the team's developmentregularity as executing unit tests is [Crispin, Miller].

Tracking the test

The teammust track on the everyday the total amount of acceptance tests written, and the amount that pass. Tracking percentages can vague certainty. For example If the squad had 200 tests yesterday and 40succeeded, nonetheless they have 210 now and 37succeed was today poorer than yesterday? No, but 20% of the tests succeed yesterday and 17.61% succeed today, basically since you had further tests. Tracking numbers can be used to defeat this problem[Crispin, Miller].

1.4.2 Process to create acceptance test

Acceptance test processed as follows:

1. The client writes stories.

2. The development squad and the client havediscussions about the story to flesh out the aspectsand build a common understanding.

3. The developer does specific investigation to understand the story better, if the acceptance test is not pure. This is an effectivedevelopment action, and the "deliverable" does nothave to be authorized by an acceptance test.

4. When the investigation is done, the developer writes a “first view” at one or more acceptance tests for thestory and authorizes it with the client. In this case, client hassufficient evidence to guess the conclusion of theremains of the story.

5. After the client and the designer have granted on the "first cut" acceptance test(s), he gives them overto business people to transcribe more tests to discover all border conditions[Rogers]

1.4.3 Relationship between Acceptance Tests and Other Kinds of Test

As we know, writing acceptance tests is typically easier than writing unit test or integration tests.This is because of the evidence that acceptance tests are transcribed, or at least want to be understood, by clients themselves. An automated acceptance test structure is possibly more difficult than that of aunit test driver, but from the test writer's point of view, things are preserved easy. When we compare the acceptance testing with unit and integration testing this provides us a better appearance on where acceptance testing fits the huge image. Unit tests commonly can be considered the tiniest scope in a project tests: unit test can be expressed the functional of a particular class, or extra minor unit in a software system. While acceptance tests are typically transcribed in a custom, area specific language that's simple forthe client to understand, unit tests are normally written in the executionlanguage of the software system[Rogers].

There are a few variances in the running of unit tests against acceptance tests. For instance, it is acceptable for an acceptance test to be unsuccessful until the story for which it was transcribed is applied. Unit tests, on the other side, must always success once integrated into the system. A great variance can also be detected in the fact that unit tests are execute in separation, while acceptance tests have noticeable side effects. The separation of unit tests guarantee that a deteriorating unit tests shouldn'tinfluence other unit tests, while a deteriorating acceptance testmight cause others to deteriorate. Integration tests are distant to acceptance tests in that they utilize on an abstract level than unit tests and validate the conduct of several portions of the system utilizing together. The major different between acceptance testing and integrationa test is that integration test can be considered as a part of the system. As well as integration tests are firmly developer-tests, and written in the natural language ofthe application [Rogers].

1.4.4 Tools for Automating Acceptance Test

Acceptance tests incorporateat anabstract level between the business logic and theuser interface or openly with the user interface. Doingacceptance testing by hand will in most cases beboring, costly and time consuming. Automation of acceptance tests might appear as a talented creativity to simple and enhance this process. The key knowledge of automated acceptance test (AAT) is to certificate requirements and needed outcome in a design that can be inevitably and continually tested. In order to catch the extreme advantage from the tests, we need tools to mechanize the real implementation of the tests. There are several of tools that used to create automated acceptance tests [Miller]. These tools will be explained in more detail in the following:

1.4.4.1 Fit and the Table –Based Technique

As we know that acceptance tests are design to test the project's performance from the client's standpoint, it must be conceivable for the client to write the tests themselves and recognize the test specification without need for prior understanding on programmingLanguages.Maybe the greatest famous execution of a table-based acceptance testingtool is Fit (Framework for Integrated Test), introduced by Ward Cunningham in 2002. [WP-FIT]. It is an open-source tool that permits clients to offer example of their application must work by writing them in a table layout.In Fit's situation, the tests are shown as normal HTML tables: the clients are welcome to usage any tool (for instance a word processor) for building the tables, as long as the figures can be transferred to HTML.

In Fit, this converting is completed by fixtures transcribed by programmers, maximum often in the real execution language of the application. A fixture is used to read the information from a table, and checks the application by utilizing the data as the input [Mugridge].

To end with, the fixture runner matches the client-set needs with the real outcomes and reports any faults bycolor-coding the table rows: red for failures, green for passed tests.The Fit framework is accessible for a several of different programming languages, for instance Java, .NET, Python, Smalltalk and C++.It must be mentioned that Fit produces specifying the test standard to be easy for non-technical people, there's a call for programmers when applying Fit: the converting of a specification table to the real application needs programmer interference. Fit-based tests are implemented by a command-line tool that takes only one HTMLfile as its input. For an extra scattered method to useFit, there is a toolnamed FitNesse. Fundamentally, FitNessedepends on the Fit, but puts the test specifications on a website from which the tests can really be performed via the press of a button. FitNesse is applied as a Wiki, which means the web pagescan be easily manipulatedvia the squad members without any precise tools other thana web browser[Mugridge].

1.4.4.2 Selenium and Web-Based Acceptance Testing

Fit and FitNesse, explainedabove, are acceptance testing toolsthat can be applied on any part of software: instead of executing inevitablyin contrast to some pre-defined interface, Fit tests should be converted to the objectivesystem case-by-case by using fixtures. By nature, these fixtures are connected tothe inner structures of the software under test, and while they don't work onthe class level such as unit tests do, for instance, the fixtures must recognize a lotabout the execution of system [Mugridge, Selenium].

In actual fact, Selenium tracks the same approach for identifying test cases as Fit: tests are well-defined in tables in an HTML page. Test outcomes are also visualized in the same methodas in Fit, by coloring the table rows depending on their outcome.The key variance between Selenium and a pure Fit-based method is that Selenium is restrict to the domain of web-based applications. Accordingly, Selenium is not auniversal intend tool for acceptance testing. The real execution of Seleniumis naturally simple: instead of demanding a server-side platform for running tests, the principal of Selenium is depend on a little HTML and JavaScript files that arelocated on the server on which the application under test exist in.

The customer-written HTML pages that cover the tests are also positioned on the server, and reaching these static HTML files byusing internet browser opens a user interface through which the tests can be performed. The test implementation happens completely inside the user's browser, with the tests calling for several pages from the originating server. According to the fact that Selenium can only be applied for testing web-based applications,it shows up with a group of prior commands that can be used to regulate the running of the tests. Nowadays, Selenium can be used for all the main internet browsers on all main operating systems [Selenium].

1.5 A Brief Introduction to Security Robustness Analysis

Security robustness analysis was introduced in earlier work [El-Attar Robustness]. Security robustness analysis is an extension to the robustness analysis technique [Rosenberg – Use case Driven]. Robustness analysis has a dual purpose: (a) it is used to complete and remove inconsistencies between the use case and domain models, thus making them more robust, and (b) its outcome is a robustness diagram which take a first-cut guess at the objects required to realize scenarios described in use cases, thus reducing the gap between the analysis and design phases. The source of the objects used are either the domain model or they are introduced by the modeler. In case objects are introduced by the modelers, these objects are then added to the domain modeling, making it more complete.

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 very relatively very simply. The notational set contains four main entities which are presented and briefly explained in Table 2 in chapter 3. Any two entities can be linked together using a solid line to loosely state that they are involved.

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 [El-Attar SEKE]. Security robustness analysis yields security robustness diagrams which introduce two new sets of objects that share similar notation to robustness diagrams (seeTable 2 in chapter 3). The first set of objects is used to model the realization of misuse and threatening behavior as described in a misuse case (see Table 3*right*in chapter 3). This set of objects is coloured black to symbolize their negation property. 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 3*left*in chapter), which are coloured 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.

1.6 Domain Model

Domain model can be described as technique used to represent the concepts of problem domain. It can be used to connect the objects in the system domain to each other. These objects can be classified into two things, physical objectsand entity objects. It explains the different entities, correlations, their features, roles, and constraints that manage the problem domain [1].

Theessential gain of a domain model is that it illustrates and limits the scope of the problem domain. The domain model can be efficiently applied to confirm and evaluate the recognizing of the problem domain among a variety of stakeholders

1.7 Problem Statement

There are several proposed methods that dependon generating test cases from different analysisartifacts, such as utilize, sequencediagram, class diagram, use case models, and activity diagram[1,2,19,20,21]. However, none of these approaches attempt to generate the securityacceptance testsin the early stage as a validation process of functional security requirements by utilizing misuse case model and security robustness diagram. This work willintroduce a novel technique to develop executable security acceptance test in the early stage from misuse case model and security robustness diagram.

1.8 Thesis Objectives

The major objective of this work is to introduce a new and simple technique to develop security acceptance tests in the early stage as a validation process of functional security requirements by utilizing misuse case model and security robustness diagram. In order to carry out this objective the following tasks will be executed.

1. Extensive survey of generating acceptance test form various artifacts will be performed.
2. A novel and simple method for creating security acceptance test from misuse model case and security robustness analysis will be introduced.
3. The proposed technique will be illustrated by casestudy.

4. The case study result will be investigated and the conclusion and future work in this field will be introduced.

1.9 Thesis Methodology

Our research methodology consists of the following phases:

Phase 1: Literature Review

In this phase, we will study the existing approaches which are used for generating acceptance tests from use cases and various artifacts.

Phase 2: To propose a new Approach

In this phase, we will propose a novel technique for generating executable security acceptance tests in the early stage validation of functional security requirements by utilizing misuse case and security robustness analysis.

Phase 3: Case Studies for Proposed Approach

In this phase, we will use case study to illustrate and investigate our proposed Methodology

Phase 4: Conclusions

The conclusions of the research and future workswill be presented.

Phase 5: Thesis Writing

Complete the thesis write-up.

1.10 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents related work ongenerating the acceptance tests from use case. Chapter 3 presents a methodology that will yield executable securityacceptance tests to validate functional security requirements stated in misuse case models. Chapter 4 presents theFaculty Search Committee system 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. Chapter 5concludes the thesis and discusses future work.

CHAPTER 2

LITERATURE REVIEW

There are several proposals which deal with problem of generating the acceptance tests from use case. In this section, we look back on work related to this issue.

2.1 Exploiting Use Cases to Derive Acceptance Tests

A number of research works have proposed various methods to develop tests based on use case descriptions [Ryser, Nebut, Briand, Basanieri]. Unlike the approach proposed in this thesis which produces *acceptance* tests, the type of tests created in these works are *system* tests [Ryser, Nebut, Briand, Basanieri]. 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.

Table 1Acceptance Testing vs. System Testing

	Acceptance Testing	**System Testing**
What and When?	***Prior to development:*** Acceptance test development is a validation	System testing is a

	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.	verificationprocess the exercises the system to determine if 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.
	After development: 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.	

Who?	***Customers***, ***end-users and business analysts (or requirements engineers)*** ideally will collaborate during the requirements engineering phase to create user acceptance tests.Customers and end-users is a crucial aspect since they possess valuable about the problem domain and the required functionalities. Business analysts apply analytical techniques to derive tests to determine if the requirements are complete, consistent and correct.	***Software developers*** will ideally create system tests to guide their development activities. ***Software testers*** will use the developed system tests to check for bugs.

The purpose and properties of acceptance testing, as outlined in Table 1, suggests 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 lifecycle 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 Produces tests that validate and verify functional *security* requirement

The TOTEM methodology utilizes various analysis artifacts, such as use case models, sequence, collaboration and class diagrams to develop test cases [Briand]. Sequence, collaboration and class diagrams are usually unavailable until after the completion of the design phase (criteria-2). The TOTEM methodology prescribes that analysis artifacts need to be heavily augmented with OCL (Object Constraint Language) expressions. The technique introduced in [Nebut] 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 an advanced skill beyond that can be expected from a customer or a BA (criteria-1).According to the duties and skills set of BAs outlined by BABOK (Business Analysts Body of Knowledge) [BABOK], 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 [Ryser], an approach named the SCENT-Method was introduced. The SCENT-Method is concerned with systematically constructingstatecharts from use case scenarios. The statecharts are detailed with pre and postconditions, data ranges, data values and non-functional requirements. Users of the SCENT-Method need to crosscheck the statechartsfor any inconsistencies, incorrectness and incompleteness. Once again, this approach is technically too demanding to be performed effectively by a customer or a BA (criteria-1). According to BABOK, the skill set required by the approach presented in this thesis should be possessed by BAs (criteria-1) [BABOK]. The only exception is the skill of performing robustness analysis. However, in previous work [El-Attar-b], 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 (criteria-4), which in turn hinders their reusability of the tests created (criteria-5). The approach presented in [Basanieri] 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 [Basanieri] prevent it from being performed during the early phases of the development process (criteria-2). An approach developed in previous work [El-Attar] satisfies criteria-1→5, but not criteria-6. The approach presented in this thesis was specifically designed to account for functional security requirements, in addition to satisfyingcriteria-1→5. There are important characteristics about the approach presented in this thesis which differentiates it from the approach in [El-Attar]. Firstly, the approach presented in [El-Attar] 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-in* to 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 [Sindre-d]. The behavioral characteristics of misuse cases significantly compound their analysis which is required to produce acceptance tests.

Secondly, the approach presented in [El-Attar] does not account for relationships between use cases such as the *include*and *extend* relationship. The approach presented in this thesis accounts for such relationships, in addition to relationships between misuse cases and use cases, namely the *threatens* and *mitigates* relationship.

CHAPTER 3

THE PROPOSED METHALODGY

In this section we will explain the proposed methodology that will yield executable security 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 Figure2. The main phases are described in greater detail in sections 3.1-3.3.

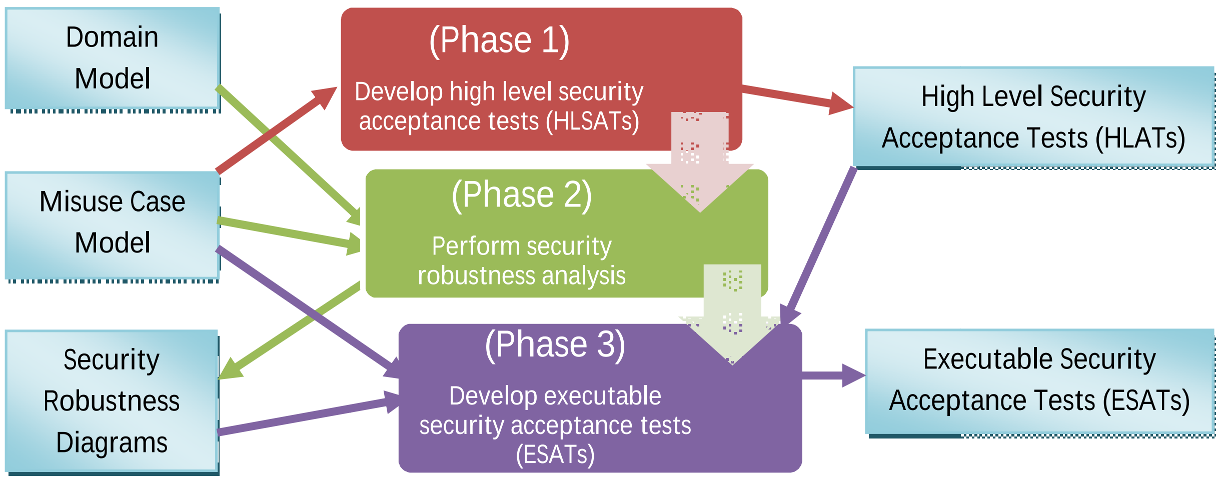

Domain Model
Misuse Case Model
Security Robustness Diagrams
(Phase 1)
Develop high level security acceptance tests (HLSATs)
(Phase 2)
Perform security robustness analysis
(Phase 3)
Develop executable security acceptance tests (ESATs)
High Level Security Acceptance Tests (HLATs)
Executable Security Acceptance Tests (ESATs)

Figure 2An overview of the proposed methodology

Phase 1: Develop security acceptance tests at a high-level based on each set of interrelated use and misuse cases (Figure 2). Use and misuse cases are considered interrelated when they form a web of associations that consists of a usage case; behavior that threatens the system when performing this usage case; 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 security 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.

3.1 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 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 describes 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 Figure 3.

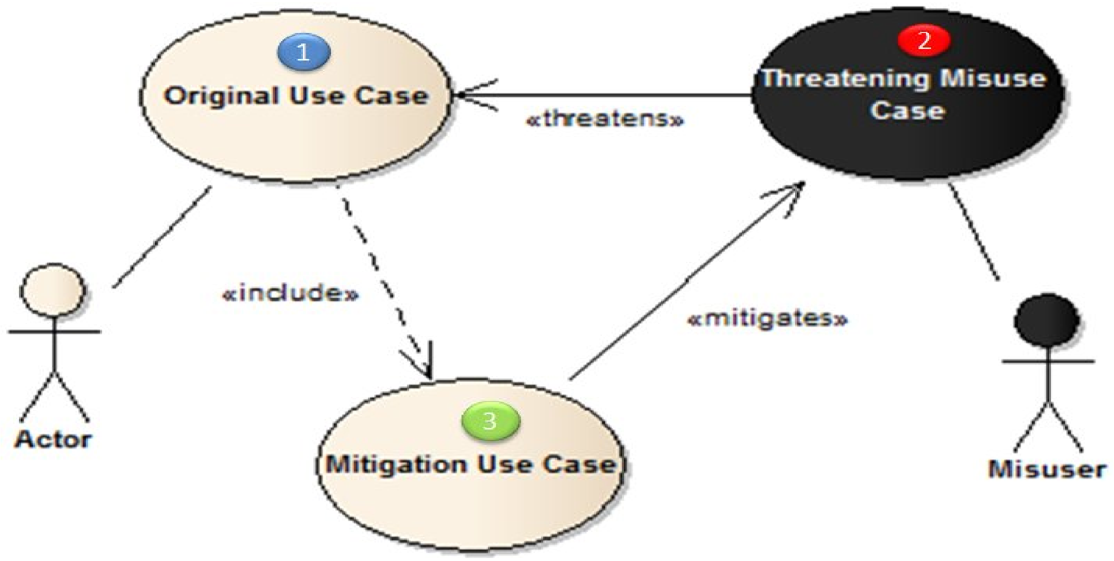


Figure 3 Usage patterns in misuse case models

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

Upon identifying the usage scenarios from Q1, it is required to identify the needed inputs, triggers and pre-conditions. Inputs and triggers are usually provided by actors, misusers, use cases, misuse cases or other systems. The pre-conditions 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.

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.

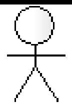
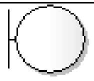
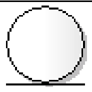
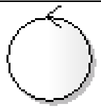
There may be cases where the above three questions are difficult to answer. This is usually symptomatic of a low quality misuse case model. Such misuse case models describe behavior that is ambiguous, too abstract or incomplete. Performing security robustness analysis (Phase 2) will attempt to remedy 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.

3.2 Phase 2: Performing security robustness analysis

Security robustness analysis was introduced in earlier work [El-Attar-d]. Security robustness analysis is an extension to the robustness analysis technique [Rosenberg]. 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 are either the domain model or they are 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.

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 simply. The notational set contains four main constructs shown in Table 2. Any two entities can be linked together using a solid line to loosely state that they are associated.

Table 2Robustness 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.

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 [El-Attar-d]. Security robustness analysis yields security robustness diagrams which introduce two new sets of objects that share similar notation to robustness diagrams (Table 2). The first set of objects is used to model the realization of misuse and threatening behavior as described in a misuse case (see Table 3*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 3*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.

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. The interactions between these objects are expected to realize "play-out" the behavior stated in the use and misuse cases. Analysis of the relevant use and misuse cases narrative should be performed in three phases. In the first phase, the narrative of the use case describing the ordinary business rules is analyzed and the set of regular objects and actors are created, using the notation shown in Table 2. Pre-existing objects in the domain model should be used to develop this initial diagram whenever possible. The first phase is concerned with developing an ordinary robustness diagram. In the second phase, the narrative of the misuse case describing the misuse behavior is analyzed and a set misuse objects and misusers are created (black graphical notation). Relevant misuse cases should be identified from the misuse case diagram. Relevant misuse cases are the ones that have a *threatens* relationship directed towards 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). In the final 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* relationship directed towards misuse cases that have a *threatens* relationship directed towards to the original use case.

Table 3Extended notation of security robustness diagrams

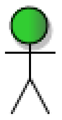
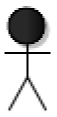
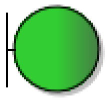
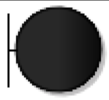
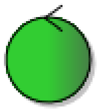
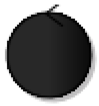
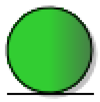
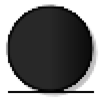
Entity	Symbol	Entity	Symbol
Mitigator		Misuser	
Mitigation Boundary		Misuse Boundary	
Mitigation Control		Misuse Control	
Mitigation Entity		Misuse Entity	

Figure 3 presents an example of a security robustness diagram. This security robustness diagram is constructed based on the analysis of the mis/use cases presented earlier in Figure 1. The security robustness diagram shown in Figure 4 illustrates how the crook restaurant owner is expected to threaten the integrity of the search results. The crook restaurant owner intercepts and reveals the search request from the customer. The crook restaurant owner then proceeds to store user preferences and execute the search himself. The crook restaurant owner then manipulates the results by removing some of their competitor's restaurants from the search results and unfairly highlighting their own restaurant. Mitigation efforts include data encryption of the search terms, double-checking of the list of restaurants retrieved and the restoration of viewing defaults.

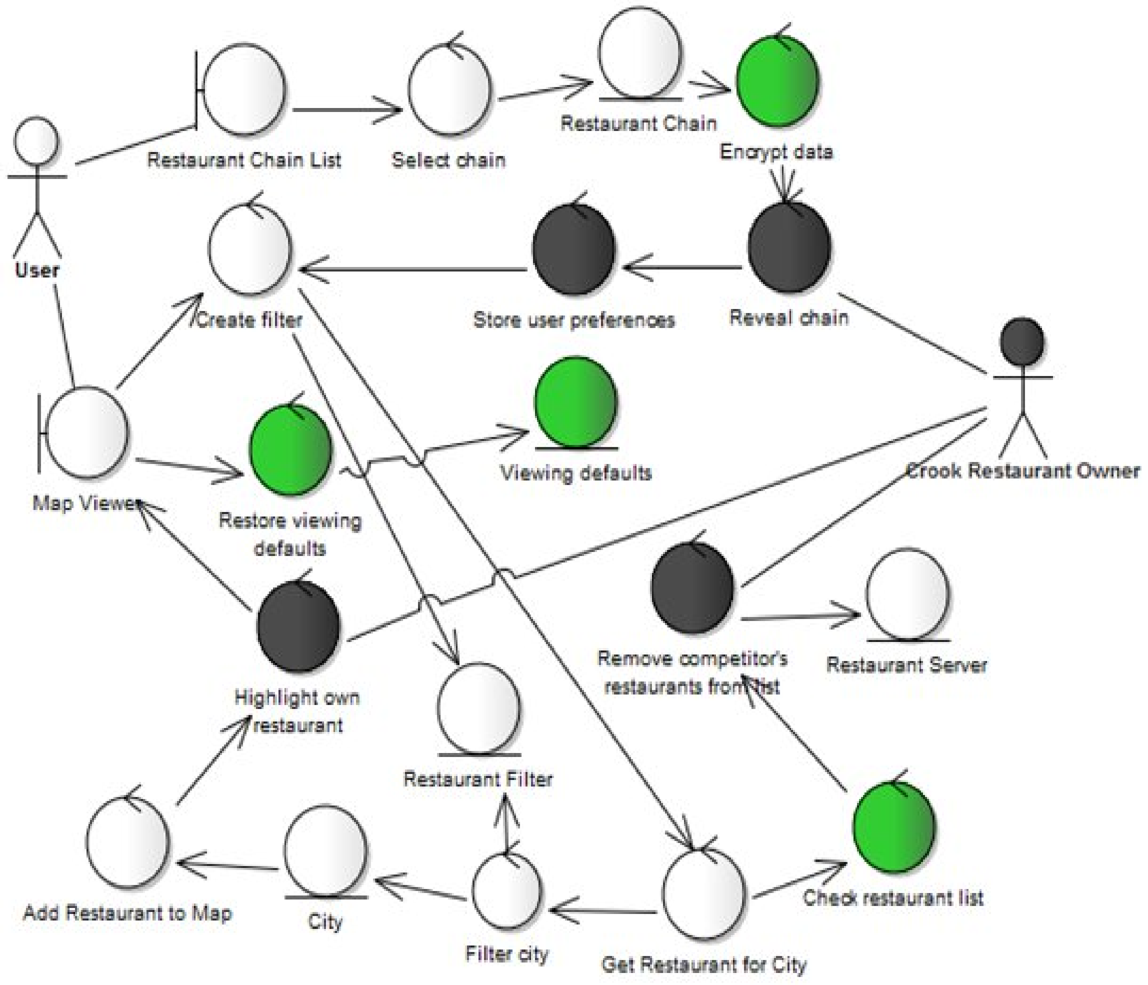


Figure 4 Example security robustness diagram

3.3 Phase 3: Developing 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 securityacceptance tests. There exist many tools than provide automation support for this phase, such as FIT/FITnesse [Mugridge] and Selenium [Selenium]. 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) ActionFixture, (b) ColumnFixture, and (c) RowFixture. 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 (expect 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 4 provides a brief explanation of each command. The subsequent fields are used to include the necessary information to execute the command.Using the running example of the restaurant system, an ActionFixture example can be shown in Figure 5.

Table 4 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.

MapViewer		
Enter	MapViewer.CurrentCity	Boston
Enter	DisplayFilter.isSmoking	False
Press	MapViewer.displayMap()	
Check	MapViewer.isMapDisplayed	True

Figure 5 Example ActionFixture

A RowFixture is used to check sets of data. The header of a RowFixture indicates the name of the data 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. Figures 6 and 7 show examples of a RowFixture and a ColumnFixture, respectively.

MapViewer.Map.DisplayedRestaurants		
Name	Address	Phone
China Pearl Restaurant	9 Tyler St, Boston, MA	(617) 426-4338
Jacob Wirth	31 Stuart St, Boston, MA	(617) 338-8586

Figure 6 Example RowFixure

MapViewer.RestaurantName			
Date	SmokingSection	GroupSize	Reservation()
30-4-2013	No	6	Confirmed
5-5-2013	Yes	2	Unavailable

Figure 7 Example ColumnFixture

3.4 Targeted Coverage by the Developed Acceptance Tests

The proposed approach aims to develop security acceptance tests for different usage scenarios based on the usage pattern shown in Figure 3. Therefore, the tests developed aims to achieve *path* coverage. As of the time of writing this thesis, there has not been any empirical evaluation conducted that provides evidence of a better test coverage criteria for mis/use cases. Therefore, path coverage was targeted as it was deemed the most appropriate given the scenario based characteristic of mis/use cases.

CHAPTER 4

CASE STUDY

4.1 The Faculty Search Committee System Case Study

This section presents the Faculty Search Committee system 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 Faculty Search Committee system was 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 openings at the Information and Computer Science (ICS) department. In general, the users of the system include any faculty member in the ICS department. 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 thesis. 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 Figure 8. The misuse case model contains 2 actors, 1 misuser, 6 use cases and 3 misuse cases. A brief description of each element in the model is provided in Table 5. The preliminary domain model of the Faculty Search Committee system is presented in Figure 9. The domain model is usually built through a brainstorming process. Table 6 provides a brief description of the classes contained in the preliminary domain model.

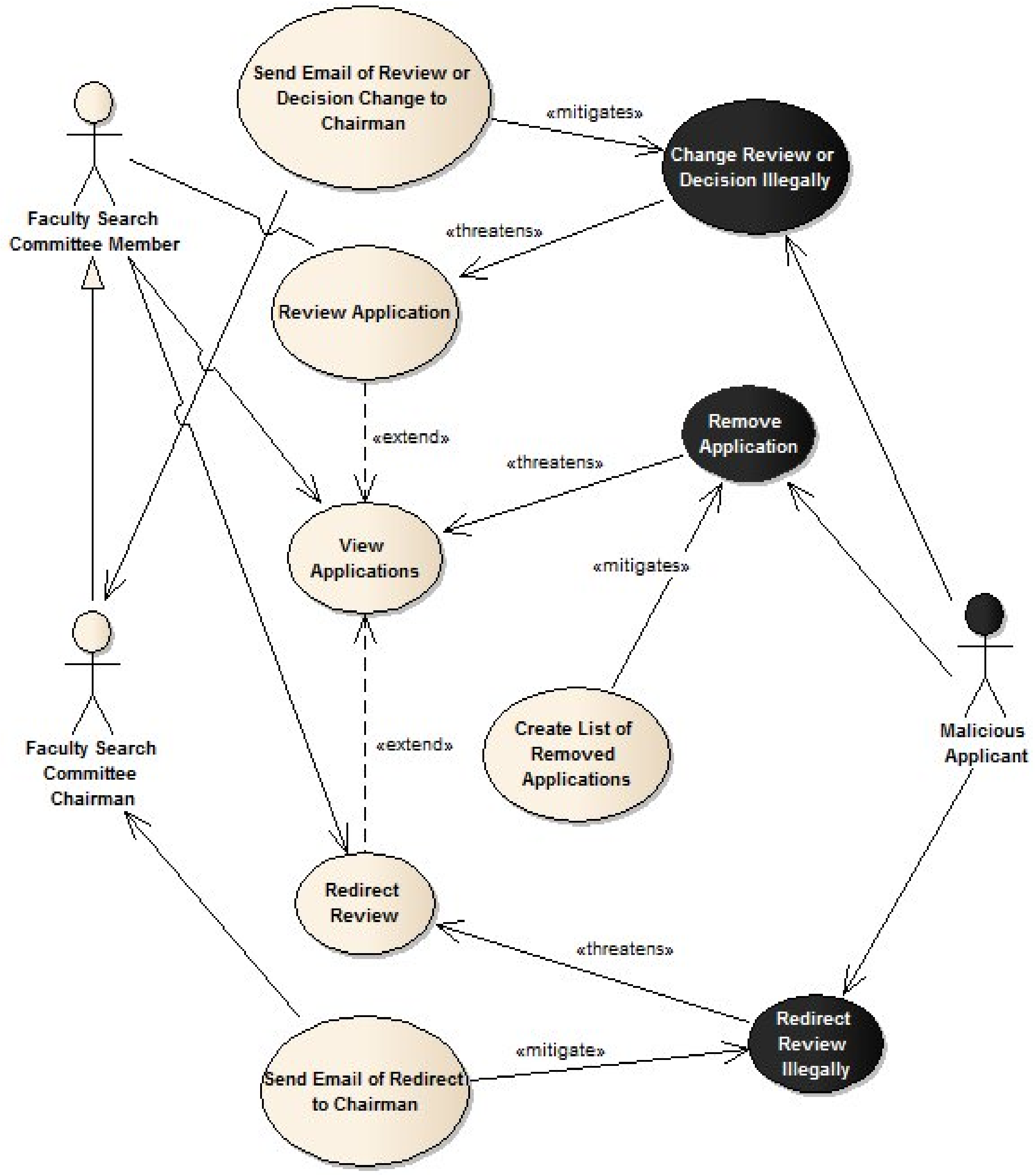

Send Email of Review or Decision Change to Chairman
«mitigates»
Change Review or Decision Illegally
Faculty Search Committee Member
«threatens»
Review Application
«extend»
Remove Application
«threatens»
View Applications
«mitigates»
Faculty Search Committee Chairman
Malicious Applicant
Create List of Removed Applications
«extend»
Redirect Review
«threatens»
Redirect Review Illegally
«mitigate»
Send Email of Redirect to Chairman

Figure 8 Misuse case model of the faculty search committee system

Table 5 Properties of the faculty search committee system misuse case model

Element	Purpose
Actors	
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.
Misusers	
Malicious Application	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.
Use Cases	
View	This use case describes the necessary behavior to view all active

Applications	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 Application	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.
Misuse Cases	
Change Review	This misuse case describes the behavior initiated by a malicious

or Decision Illegally	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.

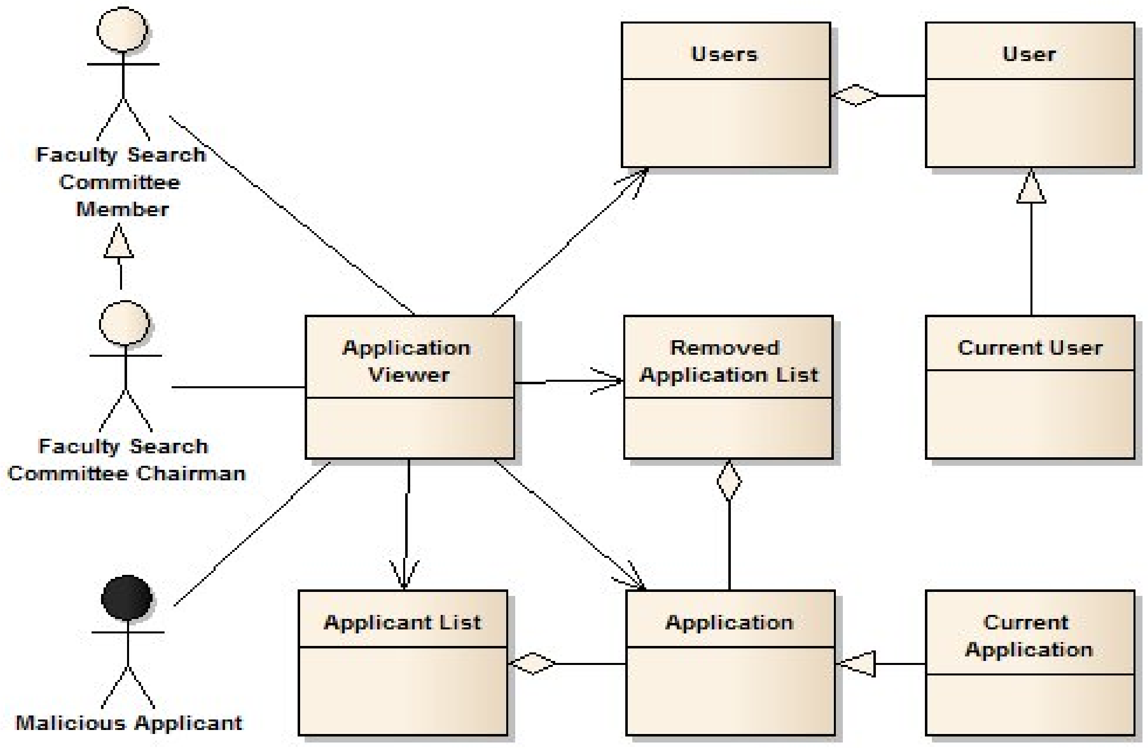


Figure 9The initial domain model of the system

Table 6 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.

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 misusage 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:

Sections 4.1.1: The behavior of the “Remove Application Illegally” misuse case and its associated use cases is presented. This behavior is analyzed in the proceeding subsections.

(Phase 1):A set of HLSATs is created to simulate the misusage behavior and test the mitigation behavior.

(Phase 2): Security robustness analysis is performed and a correspondingsecurity 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 that implement the HSLATs.

Sections 4.1.2 and 4.1.3 follow a similar structure to that of Section 4.1.1 to analyze misuse cases “Change Review Decision Illegally” and “Redirect Review Illegally”, respectively.

4.1.1 Remove Application – Misuse Case

This misuse case is initiated by a malicious applicantby 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.

Applying the proposed technique begins by examining the behavior of the involved mis/use cases. A separate HLSAT should be created for each flow between the flattened mis/use cases **(Phase 1)**. For brevity and simplicity purposes, only the "basic flow" of all mis/use cases will be considered. The following HLSAT was derived based on the behavior of the involved mis/use cases (see Table 7).

Table 7 The HLSAT for the "Remove Application" misuse case

Test ID	Description	Expected Results
Remove Application-Basic Flow	*Precondition*: Run Faculty Search Committee System *Input*: Username *Input: Password* *Input*: ID of application to be removed	*Output*: The selected application is removed but added to the list of removed applications displayed by the "Application Viewer"

In the next phase, a security robustness diagram is developed (see Figure 10) by utilizing classes already present in the initial domain model **(Phase 2)**. New objects are also introduced as necessary as a result of performing security robustness analysis. The behavior of the involved mis/use cases can be modeled by the security robustness diagram shown in Figure 10.

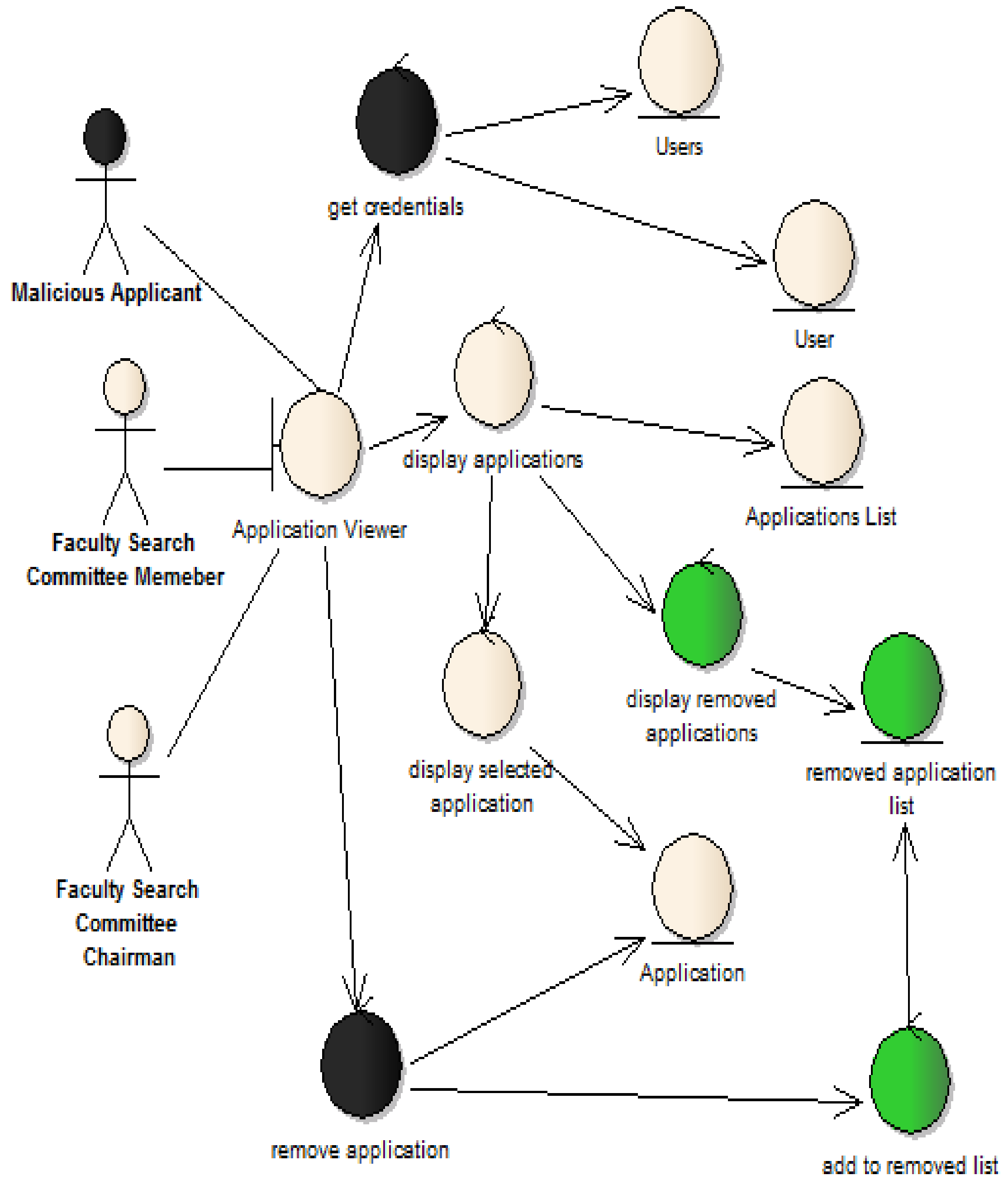


Figure 10The security robustness diagram for the "Remove Application" misuse case

The final phase is concerned with developing ESATs**(Phase 3)**. In this phase executable tests are created to realize the HLSATs previously developed (Table 8). The executable tests make use of the objects present in the corresponding security robustness diagrams

(Figure 11). The ESAT shown below consists of six fixtures. Table 8 provides a brief description of the type and purpose of each fixture

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`	*App*
Press	`ApplicationViewer.DisplayRemovedApplications()`	
Check	`ApplicationViewer.isDisplayRemovedApplications()`	*Tru*

`ApplicationViewer.RemovedApplicationList.DisplayedApplications`	
Application-1	*Username*

Figure 11 ESATs for security functionality against improper removal of an application

Table 8 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.

4.1.2 Change Review 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 or disapprove it. The HLSAT shown in Table 9 was derived based on the behavior of the involved mis/use cases **(Phase 1)**. The corresponding security robustness diagram developed is shown in Figure 12 **(Phase 2)**. Finally the ESATs developed are shown in Figure 13 (Phase 3). The ESATs shown below consists of four fixtures. Table 10 provides a brief description of the type and purpose of each fixture.

Table 9 The HLSAT for the "Change Review Decision Illegally" misuse case

Test ID	Description	Expected Results
Change Review Decision-*Basic Flow*	*Precondition*: Run Faculty Search Committee System *Input*: Username	*Output*: The selected application has its review decision changed.

	Input: Password *Input*: ID of application to be changed *Input*: New status *Input*: Chairman username *Input*: Old status	Output: The selected application has its status changed. Output: An email sent to the committee members with the details of the change. Output: The selected application has its status reverted. Postcondition: Current Status = Old Status

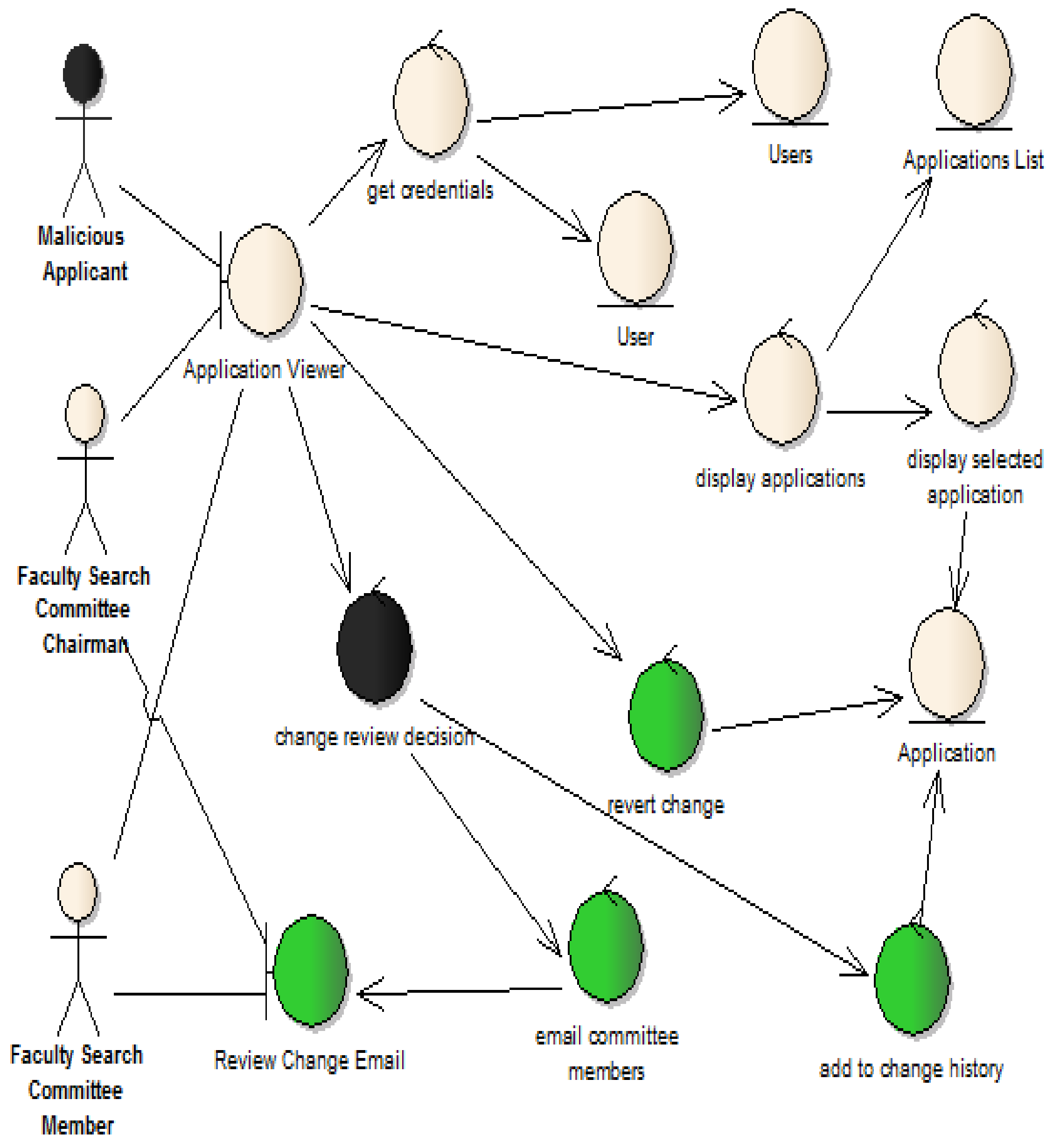


Figure 12 The security robustness diagram for the "Change Review Decision Illegally" MUC

ApplicationViewer	
Enter	ApplicationViewer.CurrentApplication

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

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`	*CommitteeChairman*
Press	`ApplicationViewer.SendChangeDecisionEmail()`	
Press	`ApplicationViewer.RejectChange()`	
Check	`Application.Status`	`Recommended`

Figure 13 ESATs for security functionality against improper overturn of a decision on an application

Table 10 ESATs for security functionality against improper overturn of a decision on an application

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 email. The fixture executes a change reject by the chairman and checks that the application's decision has not changed.

4.3 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 11 was derived based on the behavior of the involved mis/use cases **(Phase 1)**. The corresponding security robustness diagram developed is shown in Figure 14 **(Phase 2)**. Finally the EASTs developed are shown in Figure 15 (Phase 3). The ESAT shown below consists of four fixtures. Table 12 provides a brief description of the type and purpose of each fixture.

Table 11 The HLSAT for the "Remove Application" misuse case

Test ID	Description	Expected Results
Redirect Review- *Basic Flow*	*Precondition*: Run Faculty Search Committee System *Input*: Username *Input*: Password *Input*: ID of application to be redirected *Input*: New status *Input*: Chairman username *Input*: Original reviewer	Output: The selected application has its review decision changed. Output: The selected application has its reviewer changed. Output: An email sent to the committee chairman with the details of the redirect. Output: The selected application has its reviewer change reverted. Postcondition: Current Reviewer = Original Reviewer

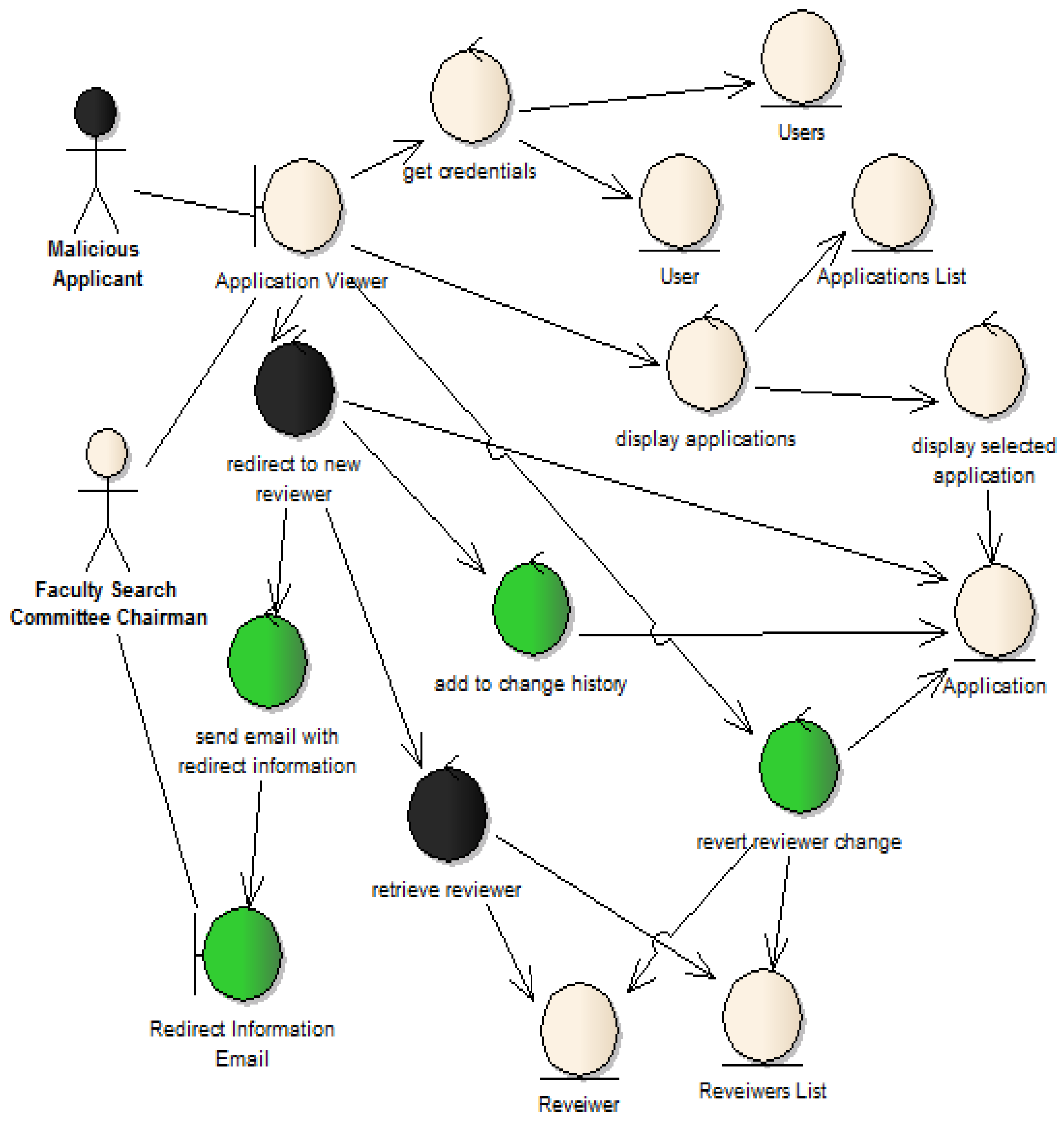


Figure 14 The security robustness diagram for the "Redirect Review Illegally" MUC

`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*

Figure 15 ESATs for security functionality against improper redirect of an application

Table 12 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 email. It simulates the chairman rejecting the change then checks that the current reviewer of the application remains as the original reviewer.

CHAPTER 5

CONCLUDINGREMARKS

5.1 Limitations

The essential unit of misuse case model is their unstructured textual descriptions. It is very hard to create totally automated methods that demand the analysis of unstructured natural language. Therefore, human estimation is involved during the application of the suggested technique. In specific, human estimation is needed to examine the textual descriptions in order to break up the stated behavior into steps, which is an essential step to define the usage scenarios, needed inputs and expected outputs. Logically, procedures that involve human judgment are subjective. The method is also reliant on the quality of the given misuse case and domain models.

Analyst inspiration. Computer crooks are certainly going to be inventive in their attempts to cheat the system. Thus analysts want to be similarly inventive to recognize the related threats in advance. 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 technique in this thesis suggests that misuse case analysis will be used early in the development process. But it might also be a clue to recreate misuse case analysis on a more specific level after the system's security defenses have been selected.This could offer a chance to test the defenses and attempt to discover weaknesses. Implementing misuse case analysis on various levels of abstraction and at many stages during the development process might raise the chance of removing such hostile surprises

5.2 Conclusion and Future Work

In the thesis 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 to an ad-hoc approach to acceptance tests creation. The feasibility of the proposed approach was demonstrated using real-world system – the Faculty Search Committee system used by the Information & Computer Science department 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 lifecycle 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 are 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 develop in earlier work [El-Attar-c]. 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 towards modifying and improving the current approach by set it to leverage the SMCD structure.

REFERENCES

[Agarwal] Agarwal, R., Venkatesh, V.: "Assessing a firm's web presence: A heuristic evaluation procedure for the measurement of usability," Inform. Syst. Res., vol. 13, no. 2, pp. 168–186, June 2002.

[Alexander] Alexander IF "Misuse cases, use cases with hostile intent ". IEEE Software 20(1):58–66. 2003.

[Alexander-b] Alexander IF"Initial industrial experience of misuse cases in trade-off analysis". In: Proceedings of the 10th anniversary IEEE international requirements engineering conference (RE'02), Essen, Germany,2002

[BABOK] International Institute of Business Analysts: Business Analysts Body of Knowledge. ttp://www.theiiba.org/AM/Template.cfm?Section=Body_of_Knowledge. Version 1.6. Last accessed February 2009.

[Basanieri] Basanieri, F., Bertolino, A., Marchetti, E.: "The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects". Proc. Fifth Int'l Conf. UML: Model Eng. Languages and Concepts and Tools, pp. 383-397, 2002.

[Briand] Briand, L., Labiche, Y.: "A UML-Based Approach to System Testing". J. Software and Systems Modeling, pp. 10-42. 2002.

[Beck] Beck K(2002) Test Driven Development: By Example. Addison Wesley

[Cockburn] Cockburn A "Writing effective use cases". Addison-Wesley, Boston,2001

[Cohn] Cohn, M.: “User Stories Applied: For Agile Software Development”. Addison Wesley, 2004.

[Crispin] Crispin, L., House, T. and Wade, C., The Need for Speed: “AutomatingAcceptance Testing in an Extreme Programming Environment”. Sec-ond International Conference on eXtreme Programming and FlexibleProcesses in Software Engineerin, pages 96_104.

[den Braber] den Braber F et al.“Model-based risk management using UML and UP. In: Proceedings of the 13th IRMA international conference”: issues and trends of information technology management in contemporary organizations (IRMA’2002), Seattle, Washington,2002

[Diallo] Diallo, M. H., J. Romero-Mariona, et al. (2006). A Comparative Evaluation of Three Approaches to SpecifyingSecurity Requirements. REFSQ'06, Luxembourg.

[Ekremsvik] Ekremsvik, S. and E. M. Tiset (2004). Misbrukstilfeller - utprøving i praksis og evaluering av teknikk. Department ofComputer and Information Science. Trondheim, Norway, NTNU. Master.

[El-Attar] El-Attar, M., Miller, J. “Developing Comprehensive Acceptance Tests from Use Cases and Robustness Diagrams”, Requirements Engineering Journal, vol. 15, no. 3, pp. 285-306, 2010.

[El-Attar-b] El-Attar, M., Elish, M. O., Mahmood, S., Miller, J., “Is In-Depth Object-Oriented Knowledge Necessary to Develop Quality Robustness Diagrams?”, Journal of Software, vol. 7, no. 11, pp. 2538-2552, 2012.

[El-Attar-c] El-Attar, M., “Towards Developing Consistent Misuse Case Models”, Journal of Systems and Software, vol. 85, no. 2, pp. 323-339, 2012.

[El-Attar-d] El-Attar, M., “Developing Precise Misuse Cases with Security Robustness Analysis”, 22nd International Conference on Software Engineering and Knowledge Engineering, San Francisco, California, USA. 2010.

[Good] Good, M., Spine, T. M., Whiteside, J., George, P.: “User-derived impact analysis as a tool for usability engineering”. in Proc. CHI’86 Human Factors in Computing Systems, 1986, pp. 241–246.

[Goodhue] Goodhue, D. L., Thompson, R. L.: “Task-technology fit and individual performance”. MIS Quart., vol. 19, no. 2, pp. 213–236, June 1995.

[Gould] Gould, J. D., Boies, S. J., Lewis, C.: Making usable, useful, productivity-enhancing computer applications. Commun. ACM, vol. 34, pp. 74–85, 1991.

[Gould-b] Gould, J. D., Conti, J., Hovanyecz, T.: “Composing letters with a simulated listening typewriter”. Commun. ACM, vol. 26, pp. 295–308, 1983.

[Gould-c] Gould, J. D., Lewis, C.: “Designing for usability: Key principles and what designers think”. Commun. ACM, vol. 28, pp. 300–311, 1985.

[Houmb] Houmb S-H et al.“Towards a UML profile for modelbased risk assessment”. In: Proceedings of the UML’2002 Satellite Workshop on Critical Systems Development. 2002.

[Holopainen] HolopainenA. “Using Misuse Cases with Non-quality Requirements”. T-76.5650 seminar in software engineering,2006

[Jürjens] Jürjens, J. “Secure Systems Development with UML”. Berlin, Springer.2004

[Jacobson] Jacobson et al.“Object-oriented software engineering: a use case driven approach”. Addison-Wesley, Reading,1992

[Kroll] Kroll, P., Kruchten, P.: “The Rational Unified Process made Easy: a Practitioner’s Guide to the RUP”. Addison-Wesley (2003)

[Kulak] Kulak, D., Guiney, E.: “Use Cases: Requirements in Context”. Addison-Wesley (2000)

[Mæhre] Mæhre, M. “Industrial experiences with misuse cases. Department of Computer and Information Science”.Trondheim, Norway, NTNU. Master,2005

[Mantei] Mantei. M. M., Teorey, T. J.: “Cost/benefit analysis for incorporating human factors in the software lifecycle”. Commun. ACM, vol. 31, pp. 428–439, 1988.

[Mugridge] Mugridge, R. and Cunningham, W. “Fit for Developing Software: Framework for Integrated Tests”. Prentice Hall, 2005.

[Miller] Miller, R. W. and Collins, C. T., “Acceptance testing. XPUniverse”, 2001, URL http://www.xpuniverse.com/2001/pdfs/Testing05.pdf

[Matulevicius] Matulevicius R, Mayer N, Heymans P.“Alignment of Misuse Cases with Security Risk Management”. In: Proceedings of the ARES 2008 Symposium on RequirementsEngineering for Information Security, pp. 1397–1404. IEEE Computer Society, Los Alamitos,2008

[Mattingly] Mattingly L, Rao H.“Writing effective use cases and introducing collaboration cases”. J Object Oriented Program 11(6):77–84,1998

[Nebut] Nebut, C., Fleurey, F., Le Traon, Y., Jézéquel, J.: “Automatic Test Generation: A Use Case Driven Approach”. IEEE Trans. on Soft. Eng. March 2006.

[Rosenberg] Rosenberg, D. & Scott, K. “UC Driven Object Modeling with UML”. Addison-Wesley, 1999.

[Rosson] Rosson, M. B., Maass, S., Kellogg, W. A.:” Designing for designers: “An analysis of design practice in the real world”. Proc. CHI + GI’87 Conf., 1987, pp. 137–141.

[Ryser] Ryser, J., Glinz, M.: “A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts”. Proc. 12 Int’l Conf. Software and Systems Eng. and Their Applications. 1999.

[Rumbaugh] Rumbaugh J.” Getting Started: Using use cases to capture requirements. Journal of Object-Oriented Programming”,1994

[Rogers] Rogers, R. O., “Acceptance testing vs. unit testing: A developer's per-spective. XP/Agile Universe”, 2004.

[Røstad] Røstad L . “An extended misuse case notation, including vulnerabilities and the insider”. REFSQ'06, Luxembourg

[Sauvé] Jacques Philippe Sauvé, Osório Lopes Abath Neto, Walfredo Cirne: “Tools and environments: EasyAccept: a tool to easily create, run and drive development with automated acceptance tests". Proceedings of the 2006 International Workshop on Automation of software test AST '06.

[Selenium] Selenium Reference Documentation. http://www.openqa.org/selenium-core/documentation.html. Last accessed Mar 21, 2013.

[Sindre] Sindre, G., Opdahl, A., Eliciting security requirements with misuse cases, Requirements Engineering Journal, vol 10, pp. 34-44 (2005)

[Sindre-b] Sindre G, Opdahl AL, Breivik GF. "Generalization/specialization as a structuring mechanism for misuse cases". In: Proceedings of the 2nd symposium on requirements engineering for information security (SREIS'02), Raleigh, North Carolina.2002

[Sindre-c] Sindre G, Opdahl AL. "Templates for misuse case description. In: Proceedings of the 7th international workshop on requirements engineering": foundation.

[Sindre-d] Sindre, G, "Mal-activity Diagrams For Capturing Attacks on Business Processes", in the Proceedings of REFSQ'07 Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 355-366, 2007.

[Schneider] Schneider G, Winters J."Applying use cases—a practical guide". Addison-Wesley,1998

[Stålhane] Stålhane, T. and G. Sindre. "A comparison of two approaches to safety analysis based on use cases", In Proc. ER'07, Auckland, 6-9 Nov 2007, Springer LNCS 4801.2007

VITAE

Name	: Akram AbduL-ghani Hezam
Nationality	: Yemeni
Date of Birth	: 12/7/1981
Tel.	: +966 597322758
	: +967 733322116

Academic Background **:**

- Received Bachelor of Teaching Computer– King Khalid University (KKU) – Abha, Saudi Arabia -, 2007 with a GPA of 4.74/5.
- Joint the Information and Computer science department as full time student at KingFahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in September 2008.
- Completed Master of science (M.S.) in Computer science from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in December 2013, with a GPA of 3.34/4.0.

- Email: almajd_2@yahoo.com, g200805800@kfupm.edu.sa