A SUPPORT TOOL FOR ELICITATION OF USER INTERFACE REQUIREMENTS

AZITA SOLTANIAN BOJNORD

DISERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE MASTER OF SOFTWARE ENGINEERING

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

March 2011
UNIVERSITI of MALAYA
ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Azita Soltanian Bojnord (I.C/Passport No: R1073255)
Registration/Matric No: WGC070024
Name of Degree: Master of Software Engineering

A Support Tool for Elicitation of User Interface Requirements

Field of Study: Requirement Engineering

I do solemnly and sincerely declare that:

1) I am the sole author/writer of this Work;
2) This Work is original;
3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya (“UM”), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate’s Signature

Date:

Subscribed and solemnly declared before,

Witness’s Signature

Date:

Name:

Designation:
Abstract

Requirements and user interface specifications are often elicited and specified separately, in most cases. The errors in requirements gathering and software specification phases lead to project failure and costly mistakes during software development. The lack of user involvement and the communication gap between the developers and the end-users are the major causes of poor requirements and user interface requirements elicitation. Likewise a bad user interface lessens the quality of software. The purpose of the research is to find a way to facilitate elicitation and specification of user interface requirements from users directly. This research reviewed the current works in this area and proposed an approach and applied it in a support tool (WEBSTUIRE) to help the developers and the users in user interface requirement elicitation.

The WEBSTUIRE inherits many features of WEBSTORM, a web-based prototype for software specification and modeling by Sagar Talekar, and expands it to elicit user interface requirements from UML interaction diagrams. WEBSTUIRE redesigns, re-develops and improves many of WEBSTORM capabilities, and adds new capabilities to capture user interface requirements. This research describes the WEBSTUIRE approach to user interface requirements elicitation and specification with combining scenario-based approaches and prototyping approaches in a multi-user environment. The resulting SRS document which is provided by the proposed tool, has to be well understood and well presented to the developers and end-users. It also provides an easy way to extract main functionalities, features, and user interface requirements of the proposed system. The tool has been compared with several related CASE tools and has been well evaluated and accepted by a group of potential developers. The list of future work is proposed to expand functionality and capability of WEBSTUIRE.
Keywords: software requirements, user interface requirements, requirements elicitation, actors, use cases, scenarios, prototyping, mock-up, CASE, UML, SRS
ACKNOWLEDGEMENT

I gratefully acknowledge my supervisor, Dr. Rodina Ahmad, for her patience, guidance, encouragement, and academic support, which has been invaluable to my Masters Research and the writing of my thesis. Especially, I appreciate her for taking her precious time to document this thesis, and providing valuable comments.

I owe my deepest gratitude to my husband Ali and my little daughter Helena accommodated me a lot during the hard work hours throughout my studies at UM and I can’t thank them enough for their love and support.

Also, I wish to express my special thank to my parents, Aghdas Tafaroji and Habib, for inculcating good educational values in me right from my childhood and for their continuous support and love; my sisters, Bita and Hoda, for giving me help, when I needed it.

Additionally, I would like to thank to my mother-in-law and my sisters-in-law for their love and support. Finally, I also thank all of my friends, I have made during the past two years on my path towards pursuing my M.S. Degree.
CHAPTER 1-INTRODUCTION .................................................................................1
  1.1 Introduction ..........................................................................................1
  1.2 Importance of Study ............................................................................2
  1.3 Problem Statement ..............................................................................2
  1.4 Research Objectives ............................................................................3
  1.5 Research Questions ............................................................................4
  1.6 Scope of Research ..............................................................................5
  1.7 Related Works .....................................................................................5
  1.8 Thesis Outline ......................................................................................6

CHAPTER 2-LITERATURE REVIEW .................................................................7
  2.1. Introduction .........................................................................................7
  2.2. Software Engineering Methodology ..................................................8
  2.3. Requirements and User Interface Requirements ...............................9
  2.4. Actors and Use Cases .........................................................................11
  2.5. Scenarios ..........................................................................................13
  2.6. Requirements Traceability Matrix .....................................................16
  2.7. Sequence Diagrams ...........................................................................17
  2.8. Scenario Screen and Mockup .............................................................19
  2.9. Navigation Map ................................................................................19
  2.10. Support Tools and Similar Work .....................................................20
  2.10.1 Telelogic’s DOORS and Scenario Plus ............................................20
  2.10.2 Enterprise Architect ...............................................................21
  2.10.3 Altova UModel 2010 ....................................................................21
  2.10.4 OSRMT .........................................................................................22
  2.10.5 Requirements Use Case Tool (RUT) ...............................................23
  2.10.6 SUCRE .........................................................................................24
  2.10.7 STORM .......................................................................................25
  2.10.8 WEBSTORM ...............................................................................26
  2.10.9 SUIP tool .....................................................................................26
  2.10.10 Scenario-based Model Driven Engineering Framework .............27
  2.11. Brief Discussion and Comparison ....................................................27
  2.12. Conclusion .......................................................................................29

CHAPTER 3-RESEARCH METHODOLOGY ..................................................30
  3.1 Research Methodology .......................................................................30

CHAPTER 4-OVERVIEW OF UI REQUIREMENTS ELICITATION USING
COMBINATION OF SCENARIOS AND PROTOTYPE .........................................34
  4.1. Introduction .......................................................................................34
  4.2. System Overview .............................................................................34
  4.3. Method Overview .............................................................................35
  4.3.1 Extracting Requirements .................................................................37
  4.3.2. Analysis and Validating Requirements ........................................39
  4.3.3. Generating Scenario and sequence diagram ..................................40
  4.3.4. Scenario Screen Creating and Analyzing .....................................43
# References

8.3. Conclusions

8.1. Objectives and Goals

8.2. Future Research Works

8.3. Conclusions

## Chapter 8

### 8.4. Future Work and Conclusions

8.1. Objectives and Goals

8.2. Future Research Works

8.3. Conclusions

References

APPENDIX A
List of Figure

FIGURE 2.1 EXAMPLE OF ACTOR STEREOTYPE ............................................................. 11
FIGURE 2.2 EXAMPLE OF USE CASE STEREOTYPE AND SYSTEM BOUNDARY .................. 11
FIGURE 2.3 EXAMPLE OF USE CASE DESCRIPTION OF SIGN UP USE CASE ...................... 12
FIGURE 2.4 EXAMPLE OF RELATIONSHIPS BETWEEN USE CASES ............................... 13
FIGURE 2.5 EXAMPLE OF A PRIMARY SCENARIO ...................................................... 14
FIGURE 2.6 EXAMPLE OF A SECONDARY SCENARIO ................................................. 15
FIGURE 2.7 SECONDARY SCENARIO: DUPLICATE SUBSCRIPTION ................................. 16
FIGURE 2.8 TRACEABILITY MATRIX EXAMPLE ....................................................... 17
FIGURE 2.9 EXAMPLE OF SEQUENCE DIAGRAM ...................................................... 18
FIGURE 2.10 SUCRE FRAMEWORK (ALSUMAIT, 2004) ............................................... 25

FIGURE 3.1 THREE PHASES PROCESS OF CONDUCTING THIS RESEARCH: .................. 33

FIGURE 4.1 SIX MAIN STEPS OF GENERATING USER INTERFACE REQUIREMENTS 
ELICITATION AND SPECIFICATION ........................................................................... 36
FIGURE 4.2 FOUR SUB-STEPS OF REQUIREMENTS EXTRACTING STEP ........................ 38
FIGURE 4.3 THREE SUB-STEPS OF REQUIREMENTS VALIDATING STEP ........................ 39
FIGURE 4.4 FOUR SUB-STEPS OF GENERATING SCENARIO AND SEQUENCE DIAGRAM STEP ...... 43
FIGURE 4.5 SUB-STEPS OF CREATING AND ANALYZING SCENARIO SCREEN STEP ............. 44
FIGURE 4.6 FIRST MOCK-UP OF BORROWBOOK USE CASE ....................................... 50
FIGURE 4.7 SECOND MOCK-UP OF BORROWBOOK USE CASE ..................................... 50
FIGURE 4.8 THIRD MOCK-UP OF BORROWBOOK USE CASE ...................................... 51

FIGURE 5.1 USE CASE DIAGRAM OF WEBSTUIRE ................................................. 56
FIGURE 5.2 OBJECT MODEL OF WEBSTUIRE ............................................................ 71

FIGURE 6.1 PACKAGE DIAGRAM OF WEBSTUIRE .................................................... 74
FIGURE 6.2 DETAILED PACKAGE DIAGRAM OF WEBSTUIRE ................................... 75
FIGURE 6.3 ENTITY RELATIONSHIP DIAGRAM OF WEBSTUIRE ................................. 76
FIGURE 6.4 MODEL VIEW CONTROLLER ................................................................. 77
FIGURE 6.5 LOG IN PAGE ............................................................................................ 78
FIGURE 6.6 SIGN UP PAGE ......................................................................................... 79
FIGURE 6.7 CREATE PROJECT PAGE ........................................................................... 80
FIGURE 6.8 REQUIREMENTS MANAGEMENT PAGE .................................................. 81
FIGURE 6.9 ACTOR MANAGEMENT PAGE ................................................................... 82
FIGURE 6.10 USE CASE MANAGEMENT PAGE .......................................................... 83
FIGURE 6.11 ASSOCIATED USE CASE TO ACTOR PAGE ............................................ 85
FIGURE 6.12 TRACEABILITY MATRIX PAGE ............................................................... 87
FIGURE 6.13 SCENARIO GENERATION PAGE ............................................................. 89
FIGURE 6.14 SEQUENCE MANAGEMENT PAGE ................................................................. 91
FIGURE 6.15 SCENARIO SCREEN GENERATION PAGE ...................................................... 93
FIGURE 6.16 NAVIGATION MAP GENERATION PAGE ....................................................... 95
FIGURE 6.17 UPLOAD MOCKUP PAGE .............................................................................. 96
FIGURE 6.18 SHOW MOCK-UP PAGE ............................................................................... 97
FIGURE 6.19 SRS REPORT PAGE .................................................................................... 99
FIGURE 6.20 CONTINUE OF SRS REPORT PAGE ............................................................ 100
FIGURE 6.21 COMMENT REPORT PAGE .......................................................................... 101
FIGURE 6.22 MOCK-UP REPORT PAGE .......................................................................... 102

FIGURE 7.1 THE ROLE OF PARTICIPANTS IN WEBSTUIRE ........................................... 112
FIGURE 7.2 AVERAGE RESULTS OF EVALUATING OFWEBSTUIRE USABILITY .............. 113
FIGURE 7.3 AVERAGE RESULTS OF EVALUATING OF WEBSTUIRE FUNCTIONALITY .... 114
List of Table

TABLE 2.1 SOME OF CASE tools with their critical criteria ........................................ 28

TABLE 4.1 MAIN requirements of library system ....................................................... 45
TABLE 4.2 ACTORS of library system ................................................................. 45
TABLE 4.3 USE case description for Borrow Book ................................................. 46
TABLE 4.4 USE case of library system with related actors ....................................... 47
TABLE 4.5 USE case of library system with related requirements ................................. 47
TABLE 4.6 SEQUENCE diagram for BorrowBook use case ........................................ 48
TABLE 4.7 FIRST scenario screen of BorrowBook .................................................... 48
TABLE 4.8 SECOND scenario screen of BorrowBook .................................................. 49
TABLE 4.9 THIRD scenario screen of BorrowBook .................................................... 49
TABLE 4.10 NAVIGATION map for BorrowBook use case .......................................... 49

TABLE 5.1 FUNCTIONAL requirements of WEBSTUIRE ........................................... 53
TABLE 5.2 NON-FUNCTIONAL requirements of WEBSTUIRE ................................ 55
TABLE 5.3 WEBSTUIRE use case for Manage requirement ........................................ 57
TABLE 5.4 WEBSTUIRE use case for Add requirement .............................................. 58
TABLE 5.5 WEBSTUIRE use case for Edit requirement ............................................. 59
TABLE 5.6 WEBSTUIRE use case for Delete requirement .......................................... 60
TABLE 5.7 WEBSTUIRE use case description for Manage use cases ............................ 60
TABLE 5.8 WEBSTUIRE use case for Create use case ................................................ 61
TABLE 5.9 WEBSTUIRE use case for Edit use case ................................................... 62
TABLE 5.10 WEBSTUIRE use case for Delete use case .............................................. 63
TABLE 5.11 WEBSTUIRE use case for Generate scenario .......................................... 64
TABLE 5.12 WEBSTUIRE use case description for Manage sequence diagram65
TABLE 5.13 WEBSTUIRE use case for Create sequence ............................................ 66
TABLE 5.14 WEBSTUIRE use case for Edit sequence ............................................... 67
TABLE 5.15 WEBSTUIRE use case for Delete sequence ............................................ 68
TABLE 5.16 WEBSTUIRE use case for Generate scenario screen ............................... 69
TABLE 5.17 WEBSTUIRE use case for Generate Navigation map .................................. 70

TABLE 6.1 SOFTWARE requirements needed for running WEBSTUIRE ....................... 104
TABLE 6.2 HARDWARE requirements needed for running WEBSTUIRE ....................... 104

TABLE 7.1 WEBSTUIRE comparison with similar CASE tools .................................. 109
CHAPTER 1-INTRODUCTION

1.1 Introduction

“Requirements engineering is not a smooth and clear cut process” (Samarasinghe, 2006). In software requirements engineering process, identifying the stakeholders and their needs is an essential issue, because software system will implement base on them. Hence, these have to be documented in a form that is amenable to review analysis, communication and subsequent implementation (Alsumait, 2004; Nuseibe, 2000). Requirements elicitation is recognized as one of the most critical stages of software development which depends on the quality of the requirements and poor execution of this phase leads to later problems in system design (Davis, 2003; Hickey, 2003). The user interface (UI) is the section of software which a user directly interacts with. The user interface of an application is often one of key factors determining its success so capturing user interface requirements couldn’t be separated from elicitation phase, because designing a great user interface of systems depends on well specified user interaction requirements. Also capturing and documenting the user interface requirements improves the quality of initial design and reduce the cost of correction of errors at the early stage.

There are various techniques for user interface requirements elicitation which provide an environment where developers can elicit and validate user interface in a systematic way. A Scenario-based and Use case approaches are major techniques which can be employed to capture requirements especially dynamic interaction of user interface requirements. Developer uses scenarios to discover user needs and to explore system functionality under both normal and exceptional situation (Alsumait, 2004).

Model notations are methods which have been used to model, elicit, and document requirements and to facilitate communication of requirements among
stakeholders (Virani, 2008). Unified Modeling language (UML) is the well-used notations based on the object oriented approach.

In recent years the development of highly interactive software systems with graphical user interfaces has become increasingly common. The acceptance of such a system depends to a large degree on the quality of its user interface. Prototyping is an excellent means for generating ideas about how a user interface can be designed, and it helps to evaluate the quality of a solution at an early stage (Bumer and et al, 1996). The prototyping models UI requirements to get the final UI design of the system.

1.2 Importance of Study

Whereas, the successful software product development depends on well-defined requirements, errors may be brought on by human-computer interaction through bad user interface (Mukasa and Kaindl, 2008). The user interface requirements are quite difficult to be understood, represented and designed. Likewise, any software support tool developed to assist this requirements engineering process is critical. So, there is a vital need to develop support tool which take into account user interface requirements in a suitable software requirements specification (SRS).

1.3 Problem Statement

This research is considered under the requirements engineering area, and it is in the field of requirements elicitation in user interfaces requirements. There are a lot of scenario and use case approaches employed in requirements elicitation but they are not appropriate for user interface requirements elicitation and are not integrated with techniques to model and prototype the user interface requirements, because these techniques require accurate definition of scenario and use case technique’s constructs to suit user interfaces, in practical aspects it needs a well defined process in order to apply
the techniques. In these techniques, the relationship between requirements and use case and scenarios is vague.

There are a number of methods for the usage of prototype also. Even though use cases and prototypes are frequently used together in software development, practical approaches combining requirements specification, scenarios, prototypes, and evolutionary system development do not exist (Mannio and Nikula, 2001). Also another problem in the requirements elicitation is the communication gap between the developers and the users to get feedback from the users. As a result the developers may not have a clear idea what kind of system and user interface they are building and the end-users are often surprised when they notice that the user interface of final product does not fulfill their expectations and does not solve the problems the system is designed for in the first place.

1.4 Research Objectives

This research examines design and development of the tool for scenario-based and prototyping-based user interface requirements elicitation. The creation and analysis of prototype and scenarios are studied via proposed tool aimed to gather and document user interface requirements.

The research project includes the following objectives:

1. To outline an approach to elicit user requirement and user interface requirements which makes use of scenario-based and prototyping techniques.

2. To represent the user interface requirements to support the proposed approach.
3. To facilitate the communication between users and developers in eliciting user interface requirements through the application of web-based techniques.

In the first step in this thesis, the researcher tries to study the problems of existing elicitation techniques and interface requirements representation to resolve existing problems. The other steps are to identify the problems of applying scenario-base and prototyping technique in the requirements elicitation projects and to find out approaches to apply these techniques in the software projects efficiently. Developing this tool leads to assess the following concerns:

- To determine a proper approach to apply scenario-based and prototyping technique together.
- To evaluate the effectiveness and efficiency of applying these integrated approach of scenario-base and prototyping techniques in a case study.
- To identify advantages and disadvantages of applying this approach to capture user interface requirements.

1.5 Research Questions

The main questions will be answered by this thesis are:

1. Does the use of the scenario-based and prototyping techniques help the process of user interface requirements elicitation and specification?

2. Is the communication between developers and users improved with the usage of combining the scenario-based and prototyping techniques?

3. Can the developers facilitate elicitation and validation of user interface requirements process with the usage of combining the scenario-based and prototyping techniques?
1.6 Scope of Research

The main areas associated for this research are:

1. Requirements elicitation
2. Scenario-based technique with UML
3. The concepts of prototyping.

The purpose of this thesis is to identify, represent and validate the user interface requirements using scenario-based technique and prototyping. Over these years, Scenario-based technique has gained great popularity through use case approach. Prototyping technique is a simple and effective technique mostly used to validate the UI requirements. This thesis focuses on elicitation of the software user interface requirements through the scenario-based technique and the prototyping technique and implementing the scenario-based technique and the prototyping-based tool for usage in requirements elicitation phase to extract user interface requirements.

1.7 Related Works

The user interface is considered as main part of any software application and user interface requirement elicitation is a process that must be considered in software development. Many approaches have recognized that it is beneficial to go beyond programming of the user interface behavior and use some kind of dynamic models for its specification and validation (Virani, 2008). Scenario-based technique is used to represent and validate user interface requirements while improve and mediate the communication between different parties involved (Alsumait, 2004). Scenario-based technique appears in prototyping (Mannio and Nikula, 2001). Many researchers found Scenario-based technique as a tool for requirement elicitation. Sutcliffe (1997), McGraw, and Harbison (1997) proposed that the use of prototypes, mock-ups, examples, scenes, and narrative descriptions of contexts could all be called scenario-based approaches.
Developers consider this technique to recognize that more important functionality of system and more important aspect of system’s user interface. Although there are some tools to apply scenario-based technique and prototyping in software development, however actually there is no specific methodology to integrate these techniques in the requirements elicitation phase. Until now, many studies in area of requirements engineering have been done on methods for applying scenario-based technique and prototype technique in the software development. Indeed, there was no specific tool to integrate these techniques in order to apply them in software development. Therefore, there is a vital need on developing appropriate software support tool which aims to assist using these techniques in a technical way.

1.8 Thesis Outline

This thesis is structured as follows:

Chapter 1 explains an introduction of this thesis and described the motivation, problem statements and objectives of this thesis.

Chapter 2 describes major context related to title of this thesis and includes an overview of the scenario-based and the prototyping techniques

Chapter 3 explains about the methodology of this research.

Chapter 4 proposes a technique to combine the scenario-based and the prototyping approaches in applying in a case study.

Chapter 5 discusses about requirements analysis of the proposed software tool.

Chapter 6 presents the design and development of the tool.

Chapter 7 provides a comparison with similar existing tools with the focus on the requirements specification and the usability evaluation of proposed tool are examined.

Finally, Chapter 8 recalls the discussion and conclusions of this thesis.
CHAPTER 2-LITERATURE REVIEW

2.1. Introduction

Over the past years, requirements engineering has received significant attention and has been recognized to be an important phase. Many studies have shown that insufficient, incompatible, or vague requirements have a critical impact on the quality of the software development (Alsumait, 2004; Bell and Thayer, 1976). The result of several studies showed the correction of errors during the requirements engineering phase cost up to 200 times less than correction during design phase (Alsumait, 2004; Bohmn, 1981; Davis, 1993; Nusseibeh and Easterbrook, 2000). Essentially, requirements engineering as an engineering regulation is concerned with all aspects of software production (Talekar, 2008). Many different technologies and tools are available to help developers solve problems arising during development of software.

The Unified Modeling Language (Booch, 1998; Fritzinger, 2006; Talekar, 2008; UML, 2009) has become a well-known standard for the software engineering field that is used to document, construct and design software systems. UML consists of various models and constructs to develop efficiently and productively the software development blueprints.

Another most difficult problem in requirements elicitation of systems, especially user interface requirements, is the communication gap between the end-users and the developers. Usage of prototypes, which present a user interface the end users can react to, is a well-known solution to narrow the communication gap between end-users and developers (Kimmond, 1995; Mannio and Nikula, 2001).
2.2. Software Engineering Methodology

A software engineering process includes activities that require very serious planning to reach the final production of the software system. It naturally is complex, diverse and creative and needs extracting detailed knowledge of the user before making decisions about features of the software system (Sommerville, 2004). Thus, user participation in software engineering is considered in order to discover their needs and points of view, validate specifications, resolve conflicts, and hence build better systems (Alsumait, 2004; Darke and Shanks, 2008; Koh and Heng 2008; Vredenburg et al., 2002). Wiegers (2003) mentioned software requirements elicitation as the most important activity, as the heart of requirements engineering, in creating a software product. Whereas end-users do not necessarily have opinions about what the software should do and just have idea about their needs, requirements engineering which includes requirements elicitation, is a very important phase to get correct and accurate requirements of the software product (Talekar, 2008). Also, the user interface which must be considered in this phase is a key factor in the final software product that influences the success or failure of software.

The next step in the process is the system specification. Specification phase precisely describes systems in terms of their functional goals and non functional constraints, and also focus on description of system in natural language rather than design. Requirements and specification are two important areas are focused by requirement management process. Gathering and documenting all of the requirements and generating specification in the end of the process provide a base for software architecture. Software architecture finds a best solution to design of the system. After the design is finished based on a given architecture, the implementation begins and the implemented code is tested and bugs are fixed and quality assessment follows to reach the requirement goals. Finally, the system is released and evolution begins. The system
must be considered to meet new changes and be allowed to do new things in many ways.

2.3. Requirements and User Interface Requirements

User interface requirements are defined as a knowledge base that clearly answers the following questions: “Who are the users of the interface? What tasks do the users perform using the interface? How does user communicate with the interface? How is the interface components presented to each user? What commands and actions can the user perform on the interface?” (Alsumaitie, 2004)

The IEEE standards (IEEE, 1984; IEEE, 1991) also define Requirements Engineering as follows:

“(1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements.

(2) The process of studying and refining system, hardware, or software requirements.”

The requirements of system are defined as (IEEE, 1984; IEEE, 1991):

“ (1) A condition or capability needed by a user to solve a problem or achieve an objective.

(2) A condition or capability that must be met or proposed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

(3) A documentation representation of a condition or capability as in (1) or (2)”

The descriptions of the services provided by the system are embedded in the requirements of a system. They describe expectations of the system from the viewpoint of the system stakeholders in clear statements. The stakeholders are people who have
any kind of interaction with the system. They have enough knowledge about the system domain but do not have exact knowledge about software development technologies and related processes of the software development. Hence defining the accurate requirements and eliminating incomplete, vague, or conflicting requirements are made difficult (Bray, 2002; Talekar, 2008). Constraints on system requirements classified them into two groups: Functional and Non-functional.

Whereas getting the right requirements of the system improves the quality and reduces the correction cost of errors in other phases of the software life cycle (Sommerville, 2004), it is necessary that requirements are written in a particular format that is readable for any stakeholders. This format is shown below and some tools follow it, as introduced by Arlow (2002):

R1: The system shall allow a user to search a book.

R2: The system shall be able to show the result of search in less than 30 seconds from the database.

R3: The system shall accept a password with length of 8 characters.

It facilitates determining the type of the requirements. The final goal of selecting ways to write and to categorize the requirements is to reach a clear, accurate, and consistent set of requirements that help the specification stage. Several diagrams and descriptions of the system have to be derived from the requirements, and it is a nontrivial task to turn natural language into a formal notation.

Structural and behavioral modeling diagrams help to specify the system and user interface of the system. The design of the use cases, the most important behavioral diagrams, is described in the next section.
2.4. Actors and Use Cases

An actor is an entity that could interact with the system directly or indirectly. An actor also represents different roles in the system which may be a human user, external hardware, or other system (UML, 2009). An actor is drawn as a named stick figure, as shown in Figure 2.1.

![Figure 2.1 Example of Actor Stereotype](image)

A use case is included in a flow of events that an actor interacts with on the system. It provides a high level view of observable behavior to someone or something outside the system. A use case diagram shows the interaction between the system and related actor to it. The considered notation to represent a use case is an ellipse. Another notation is a connecting line with an optional arrowhead showing the direction of control which is considered to use a use case (UML, 2009). Figure 2.2 shows an example of the use case diagram in which the actor “User” uses the “Enter the PIN” use case.

![Figure 2.2 Example of Use Case Stereotype and System Boundary](image)
The use case description is a detailed description about the flow of events that take place during execution of a use case from the point of view of the actor. It explains the specific sequence of events between the system and the external actors and also describes what the actor performs in the use case and how the system responds to the user’s actions. It explains the preconditions, post conditions, flow of actions, and scenarios (Figure 2.3).

<table>
<thead>
<tr>
<th>Use Case ID</th>
<th>USE CASE-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name</td>
<td>Sign up</td>
</tr>
<tr>
<td>Brief Description</td>
<td>Register a new user.</td>
</tr>
<tr>
<td>Actors</td>
<td>Developer, End user</td>
</tr>
<tr>
<td>Precondition</td>
<td>The link “New User” is clicked</td>
</tr>
</tbody>
</table>
| Action Description | 1- System displays user account form.  
2- User enters Name, Company Name, Email, Username, Password, Confirm password and User type.  
3- User clicks Sign up button.  
4- System creates user account in database and displays the message “Your account have successfully created” |
| Post Condition | User account is created and user is able to login to the system. |
| Scenario | SuccessfulCreatAccount  
DuplicateUsername |

Figure 2.3 Example of use case description of Sign Up Use Case

A system boundary is used to represent actors as being outside the system and use cases as inside the system (Figure 2.2).
There are three relationships among use cases. An ‘include’ relationship is a relationship in which one use case includes the functionality of another use case as part of their normal processing. An ‘extended’ relationship is used to specify that one use case extends the behavior of another use case. A ‘generalization’ relationship is a relationship in which one use case acts as child based on another use case as the parent (UML, 2009). Figure 2.4 shows three relationships among use cases.

![Diagram showing relationships between use cases](image)

Figure 2.4 Example of Relationships between Use Cases

### 2.5. Scenarios

Scenarios are a suitable solution to bridge the gap between the requirements and the user needs. “A scenario is a description of system behaviour which, when combined with other scenarios, should provide a more complete system description” (Alsumait, 2004). This property makes scenarios mostly perfect for requirement elicitation. In addition, “the use of scenarios improves the quality of RE process and the descriptions produced, because scenarios usually are less abstract and therefore easier to develop and communicate; especially by those whom uses with less formal training but
important domain knowledge” (Alsumait, 2004; Ben Achour et al., 1999; Haumer et al., 1999; Hertzum, 2003; Sutcliffe, 2003).

Accordingly, using the scenarios facilitates deep understanding of the requirements and answers the questions about the requirements, and creates awareness of conflict or missing requirements during the requirements elicitation phase by the developers and the end-users. Scenarios to elicit user interface requirements may be represented in different media formats such as natural language text, graphics, images, videos or designed prototypes and mock-ups. This section indicates how developers use scenarios to acquire the user interface requirements.

Scenario in UML is defined as a path of execution within a use case which follows a series of events in a particular sequence without any branching on conditions (UML, 2009). Every use case can have one or more scenarios related with it but each of them has one concrete sequence of interactions between the actors and the system. They are classified according to the path of scenarios into two categories: primary and secondary scenarios. Primary scenario includes events of main functionality of use case. Each use case has only one primary scenario, but may have one or more secondary scenarios. Figure 2.5 shows an example of a primary scenario.

<table>
<thead>
<tr>
<th>S1: SuccessfulLogIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: UC1 – LogIn</td>
</tr>
<tr>
<td>Type: Primary</td>
</tr>
<tr>
<td>Flow of Events:</td>
</tr>
<tr>
<td>1- System displays login form.</td>
</tr>
<tr>
<td>2- User enters Username and Password.</td>
</tr>
<tr>
<td>3- User clicks Login button.</td>
</tr>
<tr>
<td>4- System authenticate user from database.</td>
</tr>
<tr>
<td>5- System responds user name and password exist and valid.</td>
</tr>
<tr>
<td>6- System displays an approval message and redirect to the welcome page.</td>
</tr>
</tbody>
</table>

Figure 2.5 Example of a Primary Scenario
Likewise there are several secondary scenarios for the above use case which describes alternate situations other than successful login. Figure 2.6 shows an example of secondary scenario for LogIn use case. If there are lots of secondary scenarios, there will be a high probability that something may go wrong with system regarding that particular use case. They provide a great metric for developers, so are important to the process.

It is important that scenarios can be understood easily by stakeholders of the system. In writing scenarios, to reach this goal it is advisable not to use any technical jargon but simple words instead. In modern methods of software processes to define accurate requirements the developer can utilize the use cases and then break them into the scenarios. This provides an effective way to reach the most useful features of the system to the end users.

<table>
<thead>
<tr>
<th>S2: InValidIILogIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case: UC1 – LogIn</td>
</tr>
<tr>
<td>Type: Secondary</td>
</tr>
<tr>
<td>Flow of Events:</td>
</tr>
<tr>
<td>1- System displays login form.</td>
</tr>
<tr>
<td>2- User enters Username and Password.</td>
</tr>
<tr>
<td>3- User clicks Login button.</td>
</tr>
<tr>
<td>4- System authenticate user from database.</td>
</tr>
<tr>
<td>5- System cannot find user name and password in database.</td>
</tr>
<tr>
<td>6- System displays an approval message and display log in page again.</td>
</tr>
</tbody>
</table>

Figure 2.6 Example of a Secondary Scenario

Figure 2.7 shows another example of a secondary scenario. The user forgot to enter the user name or password so the system displays the relevant message.
## S2: EmptyLogIn

<table>
<thead>
<tr>
<th>Use Case: UC1 – LogIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Secondary</td>
</tr>
<tr>
<td>Flow of Events:</td>
</tr>
</tbody>
</table>

- 1. System displays login form.
- 2. User forgets to enter Username or Password.
- 3. User clicks Login button.
- 4. System authenticate user from database.
- 5. System cannot find user name and password in database.
- 6. System displays an approval message and display log in page again.

Figure 2.7 Secondary Scenario: Duplicate Subscription

### 2.6. Requirements Traceability Matrix

The traceability matrix is used in the specification phase to ensure certain requirements are mapped to certain use cases to provide analyses and validation of requirements. In the traceability matrix, requirements and use cases and relationship between them are figured in a tabular format as rows and columns as shown in Figure 2.8. When a requirement in the left column is related to a use case in the top row, a tick is put in the intersecting cell of the Table. The completed traceability matrix gives a view of the quality of both the use cases and requirements. Based on the traceability matrix, if a requirement is spread over too many use cases, it probably indicates that it is too complex and breaking it down in multiple requirements could be a good idea.

Another advantage of using traceability matrix is that it is based on the priorities of requirements and helps to show choice of use case to develop in iteration software development. In each iteration, the most important requirements and the use case that are mapped to them are selected for development. Figure 2.8 shows an example of traceability matrix.
### Requirements:

R1: The User shall be able to save the project.

R2: The User shall be able to open the project.

R3: The system shall be able to respond in 10 minutes.

R4: The User shall be able to delete the project.

R5: The User shall be able to edit the project.

### Use Cases:

UC1: SaveProject

UC2: OpenProject

UC3: RespondSystem

UC4: DeleteProject

UC5: EditProject

#### 2.7. Sequence Diagrams

“Sequence diagrams, one of the key UML notations, serve as a well accepted media among software developers, stakeholders, and tool developers. They express interactions between actors and other entities of a system to obtain desired behavior or services by transmitting a sequence of messages” (Virani, 2008).
A sequence diagram includes Lifelines and Message events. A Lifeline in sequence diagram has two important attributes that include the name and type of Lifeline. The type of Lifeline determines whether it is an actor or system. Each Lifeline contains zero or more Message events.

The sending event of message between Lifelines is represented in the Message event that contains several attributes. The first attribute is number for showing the sequence of message. The next attribute is the name of event, another one is receiver that shows the receiving Lifeline of the message. The last one is type of message and this includes synchronous, asynchronous, or reply.

The source Lifeline (sender) of a synchronous message is blocked until it receives a response from the target Lifeline (receiver) (Rumbaugh et al., 2004; OMG, 2010). The source Lifeline of an asynchronous message does not wait for a return from the receiver and continues executing. A reply message is where the receiver of an earlier message returns the focus of control to the sender of that message (OMG, 2010).

![Figure 2.9 Example of Sequence Diagram](image-url)
Figure 2.9 displays an example of sequence diagram. In this thesis, sequence diagrams are generated in table view. The benefit of the table view is that it is easy to understand the diagram. The drawback is that it requires a lot of space.

2.8. Scenario Screen and Mockup

Mockups are a method of prototyping user interfaces on paper or computer images. It facilitates requirements elicitation and understanding the requirements and user interface requirements by the user. However, modeling and reasoning about changes in user interface design are provided by Mockups. Mockups represent how the system works and this helps the developers and end-users to figure out what they need in order to identify their goals as requirements.

Mockups do not do function of the system; they just are used to look like the system (Virani, 2008). During developing mockups, each scenario that occurs in the system consists of a set of scenario screens. Each scenario screen consists of the user interface (UI) elements such as labels, text boxes, and buttons that clear some details about the properties and actions provided by the mockup.

2.9. Navigation Map

The layout of user interface is shown by a navigation map (Virani, 2008). Displaying the possible navigation paths among the interface elements is another benefit of using a navigation map. It enables a user to perceive an entire view of screens with the navigation between the screens. The navigation map can also be the base for creating activity. Another benefit of using navigation map is to make the sequence diagram more consistent. A user can determine missing paths and, as a result, determine missing mockups; rectifying these can lead to enhanced sequence diagrams (Virani, 2008).
2.10. Support Tools and Similar Work

Base on our search, there is no tool for user interface requirements elicitation, in this section we investigate several related CASE (Computer Aided Software Engineering) tools that are concerned with the specific areas of the software development lifecycle.

Many of these CASE tools are applied in one or more stages of software development but using multiple tools, hence raising the issue of interoperability among these CASE tools. A few CASE tools are integrated with a given software development environment, and some other tools facilitate the complexities that UML models have in text format. Lack of automation in user interface requirements elicitation and lack of web interface and the limitations of the existing tools cause significant need for the approach proposed in this research.

In the reminder of this section, some of the most important related tools and their main characteristics are described.

2.10.1 Telelogic’s DOORS and Scenario Plus

A tool for requirements traceability

DOORS (Dynamic Object Oriented Requirements System) was designed by 'Telelogic' which is under offer from IBM. DOORS is a requirements management tool that keeps track of changing user and system requirements with full traceability and also supplies test support by using Mercury Test Director (Alexander, 2003). This tool supports a requirements-driven idea by adding several specialized interfaces to the basic version. The first interface is added for software architects. The Analyst Plug is another interface that supports extra control over requirements management in the early phase of software development. The web-based interface helps developers to promote the idea
of common ownership of requirements and to control and distribute the requirement information up to a certain extent.

The Scenario Plus consists of a set of add-on tools to enable DOORS to be used for scenario-based requirements elicitation and analysis. It adds several add-on tools to enable DOORS to support scenario-based requirements elicitation and analysis with UML Use Cases and permit full traceability to requirements. The compatibility of DOORS is limited to its basic version without a web interface.

2.10.2 Enterprise Architect

**UML Tools for Business, Software and Real-time Systems Modeling**

Enterprise Architect (EA) (Enterprise Architecture, 2010) is a comprehensive UML analysis and design tool that covers software development process from requirements elicitation till analysis, design, testing and, maintenance phase. The important capabilities of EA are excellent graphical environment and modeling requirements directly. It also supports reverse engineering by integrating with IDEs. The supporting of DBMS (Data Base Management System) repository is the other feature of EA. It also has good graphics and UML diagram functionality and capability of delivering what it determines. These and other features in it make EA a powerful CASE tool, but it requires a web interface. It does requirements management to help developers build robust and maintainable software and support for creating use cases and scenarios to cover the requirement gathering phase in the graphical environment.

2.10.3 Altova UModel 2010

**A tool for software modeling and application development**

Altova UModel® 2010 (AltovaUModel, 2010) is a UML designing tool that designs application models visually in UML and generates Java, C#, or Visual Basic
.NET code and project documentation. Reverse engineering features engineer existing codes and generate UML 2 diagrams, then fine tune the designs and complete the round trip by regenerating code. UModel is a versatile UML tool that supports all UML diagrams and provides visual modeling of software design practically for any project. It is the simple, cost-effective way to draw on UML. It allows users to modify UML models and also regenerate reengineered code for developers that facilitate analysis, enhancement, or reuse of projects. Altova UModel includes a powerful reverse engineering capability to read Java, C# or VB Source code. A new feature of Altova UModel allows the user to draw UML sequence diagrams that describe the interactions between objects in application and specify the messages objects send and receive. These features make it easy to design effective applications.

Feature highlights of Altova UModel are the easy movement between code and UML models, and UML diagrams and vice versa. These capabilities allow developers, including those new to software modeling, to quickly leverage UML to enhance productivity and maximize their results. Likewise, it does not insist on the textual aspects of UML in detail and requirements. Effectively, it emphasizes human skill when it comes to working with project stakeholders to define project requirements early in the development process. It also lacks a web interface.

2.10.4 OSRMT

A tool for requirements management

The acronym OSRMT stands for open source requirements management tool. OSRMT (Smith, 2010) is a free CASE tool that is compatible with many operating systems. It can provide several capabilities such as requirements tracking, traceability, documentation and test case mapping, much like DOORS. OSRMT focuses on organizing, prioritizing and tracking requirements changes of a system in order to assist
developers in specifying the system and linking them with source code files. In order to improve the OSRMT the various features have been added to the new version enabling tools to create user groups for managing access to parts of system specification. It also has a System navigation function offering ease of access such as traceability, configuration management, system output, customization and security. OSRMT offers many places to take a query, if users have general or specific feedback. It is focused on requirements, so it lacks support for use case and scenario. It needs to improve the web-based user interface.

2.10.5 Requirements Use Case Tool (RUT)

The Requirements Use case Tool (RUT) facilities assessing the quality of use cases for developers. In addition, RUT is a web-based, multi user application that acts like a database repository for requirements developed as use cases. To ensure consistency, the tool processes all use case entry into the repository based on a standard use case template in natural language. The tool also provides a series of metrics useful for calculating information about the relationships among the captured use cases. The evaluation of use cases in RUT is performed by searching text and identifying risks. RUT integrates with Rational Rose (Rational Rose, 2010) for developing UML diagrams. Another feature of RUT is the dashboard view provided by the tool for managers that gives the ability to ascertain project status by viewing various use case metrics. It does not support other UML diagrams. It also does not export SRS documents.
2.10.6 SUCRE

SUCRE is a scenario and use case based requirements engineering framework (figure 2.10). The framework makes sure that (Alsumait, 2004):

1) ‘A consistent and complete requirement specification can be captured using scenarios and use cases,

2) The specification is a valid reflection of user requirements,

3) The derivation of early design artifacts such as low fidelity prototypes is possible.’

SUCRE framework divides the process of transforming users' requirement statements in a natural language to semi-formal specifications through a number of iterating phases.

The process in this framework includes four typical phases (Alsumait, 2004):

1) ‘Scenario analysis phase where a requirements definition and use cases can be created from the descriptions of the scenarios,

2) UCM-UI model construction phase where two types of maps are easily constructed using UCM-UI notations, CUCM and PUCM. These two maps capture a comprehensive picture of user interface and usability requirements,

3) Formally validating the UCM-UI model,

4) Early usability predictions of the UCM-UI models.’
2.10.7 STORM

STORM (Fritzinger, 2006) is a stand-alone tool for organizing requirements modeling that assists developers in the eliciting and specifying phase of the software life cycle. The main goal of STORM is to create a CASE tool capable of improved handling of text aspects of requirements specification and provide adequate functionality for keeping track of requirements, use cases, and scenarios. The most important feature of STORM is the various aspects that the tool covers of the software engineering process without relying on any other third party plug-in and add-on.

The tool also focuses on the requirements modeling and generates a clear and precise software requirements specification. The original STORM code was developed using the QT graphics library, which does not provide a web API hence cannot be used in a web-based application.
2.10.8 WEBSTORM

The WebStorm (Talekar, 2008) software is a UML-based web-enabled CASE tool that focuses on the early phases of software engineering. The WebStorm tool is a development of STORM. Some capabilities of STORM are redesigned by WebStorm. It also supports textual aspects of software specification and provides easy to use management of requirements, actors, use cases, and scenarios by improving the functionality of STORM. The Web-based user interface provides a multi-user environment and facilitates generation of Software Requirements Specification. However, it lacks support for interaction diagrams.

2.10.9 SUIP tool

A tool for automated user interface generation

SUIP is a Scenario-based User Interface Prototyping tool that proposes a new approach to the generation of UI prototypes from scenarios. “Scenarios are acquired as Collaboration Diagrams enriched with UI information. These Collaboration Diagrams are transformed into dynamic specifications of the UI objects involved in the system. Static and dynamic aspects of the UI are then derived from the obtained dynamic specifications of the UI objects (Elkoutbi et al., 2006).”

Finally, a UI prototype is generated that is embedded in a UI builder environment for further refinement. The collaboration diagrams and the UI prototype may be iteratively refined based on end-user feedback. The outcome of the overall process is a specification including the state-chart diagrams of all objects and a refined prototype of the UI.
2.10.10 Scenario-based Model Driven Engineering Framework

SDE Toolset a framework for requirement elicitation

This framework facilitates eliciting requirements. In this framework, analyzing models are provided by using consistency and completeness checking techniques. The main construction of the framework consists of a MagicDraw plug-in and a VS SDE Toolset. The output of plug-in is a UML Sequence Diagram in XML format based on a meta-model which conformed to the OMG standard. The VS SDE Toolset takes as an input the Sequence Diagram XML representation, and then parses it and displays it in tree format but it modifies the sequence diagram and displays it in tabular format. It also parses mockups and provides the ability to link mockups to scenarios of sequence diagrams to ensure consistency and completeness of requirements (Virani, 2008).

2.11. Brief Discussion and Comparison

Throughout several tools mentioned in the previous sections, there are some features and criteria that are supported by them. Some of important criteria are identified as the following:

- Requirements modeling
- Use case text input
- Automated scenario generation
- Full diagram capabilities
- Web/ Multi User Interface
- Sequence diagram in tabular format
- Automated UI screens generation
- Automated Navigation Map generation
- Mock-up and prototype generation
- End-User involvement
Table 2.1 characterizes some of the CASE tools mentioned in this section with number of criteria selected for comparison.

<table>
<thead>
<tr>
<th>CASE Tool</th>
<th>Requirements modeling</th>
<th>Use case text input</th>
<th>Automated scenario generation</th>
<th>Full diagram capabilities</th>
<th>Web/Multi User Interface</th>
<th>Sequence diagram in tabular format</th>
<th>Animated UI screen generation</th>
<th>Automated Navigation Map generation</th>
<th>Mock-up and prototype generation</th>
<th>End User involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebStorm</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>STORM</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>DOORS with Scenario Plus</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>RUT</td>
<td>×</td>
<td>√</td>
<td>x</td>
<td>×</td>
<td>×</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Enterprise Architecture</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Altova UModel</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>OSRM</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SUIP</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>x</td>
<td>×</td>
<td>√</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SDE framework</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>x</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td>x</td>
</tr>
<tr>
<td>SUCRE framework</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>x</td>
<td>x</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

However, most of these tools did not perform any process to elicit user interface requirements and to generate prototype and mockup. In fact, user involvement is not supported through these tools. Eliciting user interface requirements should be done by the developers themselves using these tools, without any suggested scenario screen from their side.
2.12. Conclusion

In this chapter, some of the concepts related to the scenario-based, UML techniques, and prototype based techniques are reviewed. Also, major findings of eliciting user interface requirements using these techniques are mentioned. Unfortunately, few concrete recommendations exist about how to combine scenario-based techniques and prototype techniques for user interface requirements elicitation, and even less tool support is available. The purpose of this chapter is to find existing knowledge associated with the uses of scenario-based and prototyping techniques. Reviewing current tools indicated that, researchers used scenario-based and prototyping techniques in different project efforts. However, the main goal of existing project on using scenario-based and prototyping is to gain the best focus on user interface requirements that finally use the software product.
CHAPTER3-RESEARCH METHODOLOGY

3.1 Research Methodology

This thesis plans to develop software support tool to apply combination of the scenario-based and prototyping techniques in requirements elicitation phase of software development. In this chapter, we discuss about three different phases of this research.

We show three Phases of this research structure in figure 3.1.

This research structure includes the following phases:

1- Initial Phase

Three steps of this phase include the following:

1) Literature Review

During this step, the researcher focus on wide review of existing related literatures to propose methods which can be useful in this research. At the end of this step, the researcher can gather most significant requirements and documents needed to progress in this research.

2) Evaluating the current support tools to elicit requirements and prototype user interface requirements specifies.

In this step, features of current support tools related to requirements elicitation and prototyping user interfaces requirements are studied and reviewed and also relevant characteristic of these tools need to design software tool to apply scenario-based and prototyping in user interface requirements elicitation are investigated. Steps one and two of this phase are done parallel.

3) Knowledge gap

After completing the earlier steps of this phase, the researcher can identify knowledge gap in term of gathered information in the earlier
stages. Likewise, researcher can access relevant resources and gather related important requirements needed for conducting this research.

The outputs of this phase are literature review chapter, current finding of the research, and information needed for developing tool.

2- Development Phase

This phase is divided into two steps includes following:

1) Apply the Proposed Technique in Case Study

This step helps to enhance techniques needed to design software tool to apply scenario-based and prototyping techniques in UI requirements elicitation. By applying proposed technique in the case study, the researcher can demonstrate solution to resolve the problem statement and to investigate selected method in order to develop specific software support tool.

2) Tool Development

During this step, the researcher should apply combination of scenario-based and prototyping technique which is studied in a case study in earlier step in software project and find ways to design and develop software support tool by using selected software development methodology.

The outputs of this phase are design and implementation of the tool.

3- Evaluation Phase

Finally, the researcher compares the tool developed in this research with similar CASE tools to find both similarities and dissimilarities between them. At last, the researcher evaluates usability and functionality of tool. In order to evaluate the tool developed in this research, a questionnaire form which considers some important critical functionality and usability features associated with it was designed. Then
questionnaire forms are filled up by users who used the tool developed in this research to evaluate it. Finally, the suggestions and recommendations of users in questionnaire were reflected in diagrams which were based on them. These questionnaires distributed to the ten users from different majors of postgraduate students in the faculty of Computer Science and Information Technology in UM who work as software developers at several companies. The outputs of this phase are based on evaluation tool and future work.
Initial phase

- Literature Review
  - Study all available related resources to this research.
  - Review of current findings of the topic.
  - Conceptual analysis.
  - Gather all relevant information

- Evaluating the current support tool to elicit requirements and prototype user interface requirements specifies
  - Study and investigate characteristic of existing tools

Knowledge gap
- Communication gap between user and developer
- How to apply techniques to facilitate process

Development Phase

- Review of Proposed Technique in Case Study
  - Enhanced techniques

  Tool development
  - Design of the tool.
  - Implementation of the tool.

Evaluation Phase

- Comparison with other related tools and evaluate tool
  - Compares existing tools with tool developed to find both similarities and dissimilarities between them.
  - Evaluates usability of tool.

Figure 3.1 Three phases process of conducting this research.
CHAPTER 4-OVERVIEW OF UI REQUIREMENTS ELICITATION USING COMBINATION OF SCENARIOS AND PROTOTYPE

4.1. Introduction

In our research, we investigated the possibility of using the combination of the scenario and prototyping in order to support the user interface requirements in the WEB-based Support Tool for the User Interface Requirement Elicitation, or WEBSTUIRE for short, which is proposed to be a general scope and web-based tool to help the software developer manage requirements and elicit user interface requirements. Use case and sequence diagram are scenario-based approaches that describe functionality and behavior of the complex system and using of a visual prototype assists in understanding how the software will work and what the user interface will look like. In other words it assists the developer in eliciting user interface requirements before any real code is written. This also will allow the developers to create a software product which performs more closely to the business requirements.

4.2. System Overview

The main feature of WEBSTUIRE is to assist developers in eliciting and analyzing user interface requirements by using a combination of scenario-based and prototyping technique. WEBSTUIRE also facilitates communication between developer and end-users in a web-based environment and will help developers to gather user interface requirements and improve the user interface of system being developed based on end-user’s feedback.

Some of the core functionalities of WEBSTUIRE are summed up as follows:

- WEBSTUIRE can facilitate the process of eliciting user requirements and UI requirements.
WEBSTUIRE can propose some main user interface requirements based on analysis of behavior model (sequence diagram and mock-ups).

WEBSTUIRE can provide an easy way to create SRS which includes user interface requirements based on user’s feedback of system.

4.3. Method Overview

The main goal of combination of Scenario-Based Model and Prototyping technique is to ease developer’s communication with users and improve the quality of software by extracting user interface requirements. This section discusses the process for user interface requirement extracting linking scenarios in UML models with User Interface prototypes. This process is illustrated with a case study and then by using this method, developers can elicit user interface requirements in a particular software system. This process involves six activities to gain a UI prototype from scenarios and to generate a software requirements specification of an application which include user interface requirements. The main requirements of system are acquired in textual form. Scenarios are gained in the form of UML use case and sequence diagrams are enriched with UI information. A requirements traceability matrix maps use case to requirements, and ensures important requirements of system are considered during system development. A scenario screen determines screens and elements of user interface in each screen for each scenario. A navigation map which is shown in the table is built of scenario screens. The set of obtained specification allows for the generation of a mock-up, which is embedded in a UI builder environment for further refinement. The sequence diagrams and mock-ups and other features may be refined based on user feedback. The result of overall process is a specification of user requirements and user interface requirements that validation of which has been checked by users.
In this thesis, we focus on the User Interface requirements elicitation and specification process. Figure 4.1 shows the stage of this process as detailed below:

1) Extracting Requirements: the main requirements of the system are determined and then the use cases and the actors defined and the use case description generated based on the main requirements of the system.

2) Analysis and Validating Requirements: the actors associated to the use cases and the main requirements of the system associated to the use cases; the result of this activity is shown in a traceability matrix.

3) Generating Scenario and Sequence diagram: the developer determines scenarios for each use case and then he/she elaborates the sequence diagrams annotated with UI information. Developer specifies its types as well as user interface information relating to its interactive message.
4) Creating and Analyzing Scenario Screen: the developer integrates related steps of the sequence diagrams in a special screen and creates a navigation map.

5) Generating Mock-up and Navigation map: the developer generates mock-ups based on the elements of the scenario screens and then shows the mock-ups based on the navigation map of each scenario.

6) Generate user interface requirements specification: SRS document contains everything available in a WEBSTUIRE project and also user interface requirements in the user readable format.

We will illustrate this process in detail and later we apply it in an example.

4.3.1. Extracting Requirements

This activity starts by defining the information about the existing system being specified, and then the analyst first records the main requirements of the system. In addition, the name and description of the requirements and other classification aspects of the requirements such as functional or non-functional and priority level are stored in this step and the requirements are written in a particular format of requirements writing. Likewise the capability of the editing requirements is an essential part that is considered in this step because this feature provides reviewing and updating of requirements to get correct and clear system requirements and ensures that the correct problem is being addressed by system (Lauesen, 2007).

Hence actors will be specified and the use cases will be generated based on analysis of the main requirements at the end of the activity. So the actor management is next logical steps to specify the software system under development. End users of the system being developed can be classified into categories or roles as actors. End users in a particular role can interact with the system and can also inherit the use cases from
other actors in the system. This feature helps manageability of the use cases by reusing existing use cases.

There is also a specific approach to create, display, and store the use cases and their steps. It supports reusability for use case by inheriting and extending features of use cases. Also “if-else” and “while” loops can be used to provide logical use case constructs which allow user to embed control structures in the use case steps. Use case generation is being facilitated for users by considering these features which give a clear picture to the designer and are really useful during the implementation phase as they take system developers one step closer to the pseudo-code. Xu (2004) has cited that robust use case generation leads to the creation of meaningful scenarios based on the use case paths, which will be discussed next.

A detailed analysis is done to identify use case and to define pre and post conditions. Figure 4.2 shows the sequence of steps of this activity.

![Figure 4.2 Four sub-steps of Requirements Extracting Step](image-url)
4.3.2. Analysis and Validating Requirements

The most important steps in this activity are as follows:

1) Associated actors and the use cases of the system that are being developed together to validate the actors and the use cases.

2) Associated requirements and the use cases of the system that are being developed together to validate the requirements and the use cases.

3) Generate a traceability matrix between the requirements and the use cases.

An accurate requirements analysis is an essential activity in any software development process because the success and failure in a project strongly depends on the quality of the requirements and how to select them. Therefore, a mapping between requirements and use cases can assist developers in understanding the most important requirements and to make decisions on the important aspect of the system being developed based on these requirements.

A requirements traceability matrix maps the use cases to the requirements of the system, and makes it easier to prioritize the most important system features developed during system developments and also helps to validate system requirements. The traceability matrix is presented in tabular format with the use cases for columns and the requirements for rows. Figure 4.3 shows the sequence of the steps of this activity.

![Diagram showing the sequence of steps]

Figure 4.3 Three sub-steps of Requirements validating Step
4.3.3. Generating Scenario and sequence diagram

Generating scenarios is one of the important features in this process. Scenarios are considered as the execution part of the system which happens during interaction with the system being developed. Therefore, they play an important role in software projects, and also based on how many of them are implemented, developer can estimate the overall progress of the software development lifecycle.

However, the right scenario generation is very important because of the reasons explained earlier. Two types of scenarios are considered: normal scenarios (primary scenarios), which are executed in normal situations, and exceptional scenarios (secondary scenarios) executed in the case of errors and abnormal situations. In this activity scenarios are generated automatically based on selecting the use case events from the use case description; this feature avoids human errors during scenario generation.

Developers can extract scenarios related to each use case from steps of the use case description based on the method explained earlier. So, for example there are several scenarios based on events of the use case description: Scenario 1, Scenario 2 etc.

Based on steps of use case description S1 includes several steps of the use case description and S2 includes other selected steps.

After scenario generation, the developer elaborates a number of the sequence diagrams annotated with UI information and also considers each scenario of given use case in sequence diagram generation.

An interactive message is defined as a message in a Sequence Diagram that is sent to an object. “For UI elements generation purposes, two user-defined constraints related with interactive messages are proposed. There are two standard constraints for message in UML: Vote and broadcast. The vote constraint restricts a collection of
return messages, and the broadcast constraint specifies that the constrained messages are not invoked in any particular order (Elkoutbi et al., 2006)."

The additional constraints for interactive messages include the following two constraints: inputData, outputData.

Once the analyst has specified the UI constraints of the messages in the Sequence Diagram under consideration, this information is used to determine the corresponding widgets that will appear in the UI prototype.

“The inputData constraint indicates that the corresponding message holds information input from the user. The outputData constraint specifies that the corresponding message carries information for display (Elkoutbi et al., 2006).”

Once the analyst has specified the UI constraints of the messages in the Sequence Diagram under consideration, this information is used to determine the corresponding widgets that will appear in the UI prototype. “Widget generation adheres to a list of rules (WG rules), which is based on the terminology, heuristics and recommendations found in the literature (IBM, 1991; Sun Microsystems, Inc. and AT&T, 1990; Elkoutbi et al, 2006) and which includes the following ten items:

(WG1a) An enabled textfield widget (TEX) is generated in case of an inputData constraint with a dependency to an attribute of type String, Real, or Integer, e.g., enter password(){inputData(Account.password)}.

(WG2a) A group of enabled radio button widgets (RAD) are generated in case of an inputData constraint with a dependency to an attribute of type Enumeration having a size less than or equal to 6, e.g., enter operation(){inputData(Transaction.kind)}.

(WG3a) An enabled list widget is generated in case of an inputData constraint with a dependent attribute of type Enumeration having a size greater than 6 or with a dependent attribute of type collection.
(WG4a) An enabled table widget is generated in case of an inputData constraint with multiple dependent attributes.

(WG5) A button widget (BUT) is generated for an inputData constraint with a method as dependency, e.g., insert card (pin) {inputData(ATM.insert card)}.

(WG1b) A disabled textfield widget (TEX) is generated for an outputData constraint with a dependency to an attribute of type String, Real, or Integer.

(WG2b) A group of disabled radio buttons widgets are generated in case of an outputData constraint with a dependency to an attribute of type Enumeration having a size less than or equal to 6.

(WG3b) A disabled list widget is generated in case of an outputData constraint with a dependent attribute of type Enumeration having a size greater than 6 or with a dependent attribute of type collection.

(WG4b) A disabled table widget is generated in case of an outputData constraint with multiple dependent attributes.

(WG6) A label widget (LAB) is generated for an outputData constraint with no dependent attribute, e.g., display error() {outputData(“Insufficient funds”)}. Note that for outputData constraints, we choose to generate disabled widgets, in order to comply with the initial specifications. In case of generating enabled widgets, the object behavior would in general comprise more than specified in the initial scenarios.

Furthermore, note that the above rule base may be modified to encompass particular application domains and UI styles.

We draw a sequence diagram that includes several scenarios and apply widget rules to them.
Figure 4.4 shows the steps of this activity.

```
Step 3.1: Generate scenario for each use case by selecting related steps of use case description

Step 3.2: Generate sequence diagram for each use case and related scenarios.

Step 3.2.1: Identify lifelines of sequence diagram

Step 3.2.2: Select type of message

Step 3.2.3: Select sender of message

Step 3.2.4: Define Message

Step 3.2.5: Select receiver of message

Step 3.2.6: Select constraint of message (Input, Output)

Step 3.2.6: Select widget
```

Figure 4.4 sub-steps of generating scenario and sequence diagram Step

4.3.4. Scenario Screen Creating and Analyzing

Now in this step, the developer considers sequence diagrams enriched with user interface elements and then selects steps of sequence diagram that can be collected in one screen to create scenario screens. After all of the screens are specified for a use case, the developer can make a navigation table for each scenario of the use case. Figure 4.5 shows the steps of this activity.
4.3.5. Generating Mock-up and Navigation map

In this step, the developer generates mock-up based on scenario screens generated in the last step by a mock-up builder and shows mockups for each scenario based on the navigation map table. Therefore this helps users to understand elements of UI and also learn workflow of scenarios and analysis of UI requirements.

4.3.6. Generate User Interface Requirements Specification

The User Interface Requirements Specification document contains everything available for a system being developed in a WEBSTUIRE project in Microsoft Word and PDF format. Actors, use cases, scenarios, sequence diagram and navigation table are all presented in a tabular format saving all the details.

The most important feature of this approach is that users can give their comments on each step of this process and the developer can refine them based on user feedback.
4.4 Case Study

Now, in this section we apply the explained approach in library system to illustrate how this process works in detail.

4.4.1. Extracting Requirements in Library System

The main requirements of library system are elicited in Table 4.1.

Table 4.1 Main requirements of library system.

<table>
<thead>
<tr>
<th>Requirements#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>The library system shall allow Librarian and borrowers to login to library system.</td>
</tr>
<tr>
<td>R2</td>
<td>The library system shall allow borrowers to borrow a copy of book.</td>
</tr>
<tr>
<td>R3</td>
<td>The library system shall allow Librarians and borrowers to search for books by author, year or a combination of these.</td>
</tr>
<tr>
<td>R4</td>
<td>The library system shall allow User to list the books borrowed from library.</td>
</tr>
<tr>
<td>R5</td>
<td>The library system shall allow User to view the status of book.</td>
</tr>
<tr>
<td>R6</td>
<td>Library system shall allow borrowers to return borrowed book</td>
</tr>
<tr>
<td>R7</td>
<td>Library system shall allow Librarians to extend due date for a loan upon the request of borrower.</td>
</tr>
</tbody>
</table>

Actors in library system are identified based on main requirements of library system. This is shown in table 4.2.

Table 4.2 Actors of library system.

<table>
<thead>
<tr>
<th>Actors #</th>
<th>Actor name in Library system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor1</td>
<td>Borrower</td>
</tr>
<tr>
<td>Actor2</td>
<td>Librarian</td>
</tr>
</tbody>
</table>

The analyst creates use case supported main requirements of system and defines use case description for each Use Case. The main use cases for library system are following:
UC1-Log In To System
UC2-Check Book Status
UC3-Search Book
UC4-Borrow Book
UC5-Return Book
UC6-List Borrowed Books
UC7-Extend Due date

Table 4.3 shows use case description for use case of Borrow Book.

Table 4.3 Use Case description for Borrow Book

<table>
<thead>
<tr>
<th>Name of Use Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borrow Book, UC4</td>
<td>Allows borrower to borrow book</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pre-Condition</th>
<th>Librarian must log in to system</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>1</td>
<td>Librarian enter ID of membership’s card</td>
</tr>
<tr>
<td>If</td>
<td>2.1</td>
<td>Validation successful</td>
</tr>
<tr>
<td>Basic</td>
<td>2.2</td>
<td>Librarian enter ISBN and CopyNo of book</td>
</tr>
<tr>
<td>Basic</td>
<td>2.3</td>
<td>System updates the circulation records and issues due date</td>
</tr>
<tr>
<td>Continue</td>
<td>2.4</td>
<td>System show record of borrower</td>
</tr>
<tr>
<td>Else</td>
<td>2.5</td>
<td>Show message ”Deny Request”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-Condition</th>
<th>Borrower borrows book</th>
</tr>
</thead>
</table>

4.4.2. Analysis and validating Requirements in Library System

In this phase the developer associate the actors with the use case. This helps to validate the actors and the use cases defined in system (Table 4.4). After that the developer associated the requirements with the use cases and then traceability matrix is generated to validate the requirements and use cases that are defined in library system.

Table 4.5 shows the use cases are associated with requirements for library system.
Table 4.4 Use Case of library system with related Actors

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Actors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login to System</td>
<td>Actor1, Actor2</td>
</tr>
<tr>
<td>Search Book</td>
<td>Actor1, Actor2</td>
</tr>
<tr>
<td>Check Book Status</td>
<td>Actor2</td>
</tr>
<tr>
<td>Borrow Book</td>
<td>Actor1, Actor2</td>
</tr>
<tr>
<td>Return Book</td>
<td>Actor1, Actor2</td>
</tr>
<tr>
<td>List Borrowed Books</td>
<td>Actor1, Actor2</td>
</tr>
<tr>
<td>Extend Due date</td>
<td>Actor2</td>
</tr>
</tbody>
</table>

Table 4.5 Use Case of library system with related requirements

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login to System</td>
<td>R1</td>
</tr>
<tr>
<td>Search Book</td>
<td>R3</td>
</tr>
<tr>
<td>Check Book Status</td>
<td>R5</td>
</tr>
<tr>
<td>Borrow Book</td>
<td>R2</td>
</tr>
<tr>
<td>Return Book</td>
<td>R6</td>
</tr>
<tr>
<td>List Borrowed Books</td>
<td>R4</td>
</tr>
<tr>
<td>Extend Due date</td>
<td>R7</td>
</tr>
</tbody>
</table>

4.4.3. Generating Scenario and Sequence diagram

For BorrowBook use case, there are two scenarios based on events of use case description defined in previous activity:

- S1: the status of borrower’s ID is valid
- S2: the status of borrower’s ID is not valid

Based on steps of use case description S1 includes steps 1, 2.1, 2.2, 2.3, 2.4 and S2 includes steps 1, 2.5.

After scenario generation, the developer elaborates a number of sequence diagrams annotated with UI information. In sequence diagram generation, developer considers all of scenario in related use case, too (Table 4.6).
Table 4.6 Sequence diagram for BorrowBook use case.

<table>
<thead>
<tr>
<th>Step</th>
<th>Type of Message</th>
<th>Sender</th>
<th>Message</th>
<th>Receiver</th>
<th>Widget</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Synchronous</td>
<td>Librarian</td>
<td>Enter-ID()</td>
<td>Library system</td>
<td>Text</td>
<td>Input</td>
</tr>
<tr>
<td>2</td>
<td>Synchronous</td>
<td>Library system</td>
<td>OK=Verify(ID)</td>
<td>Borrower</td>
<td>Button</td>
<td>Input</td>
</tr>
<tr>
<td>3</td>
<td>Reply</td>
<td>Borrower</td>
<td>OK=.T.</td>
<td>Library system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Enter-ISBN()</td>
<td>Librarian</td>
<td>Text</td>
<td>Input</td>
</tr>
<tr>
<td>3.2</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Enter-CopyNo()</td>
<td>Librarian</td>
<td>Text</td>
<td>Input</td>
</tr>
<tr>
<td>3.3</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Create Record()</td>
<td>Loan</td>
<td>Button</td>
<td>Input</td>
</tr>
<tr>
<td>3.4</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Show Record()</td>
<td>Loan</td>
<td>Table</td>
<td>Output</td>
</tr>
<tr>
<td>4</td>
<td>Reply</td>
<td>Borrower</td>
<td>OK=.F.</td>
<td>Library System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Asynchronous</td>
<td>Library System</td>
<td>Show message (Deny Request)</td>
<td>Library System</td>
<td>Label</td>
<td>Output</td>
</tr>
</tbody>
</table>

4.4.4 Creating Scenario Screen and Navigation map in Library system

In this stage, steps of sequence diagram that could integrate together in a screen are selected to create a scenario screen. For example, screen named EnterScreen includes steps 1 and 2 of sequence diagram of BorrowBook (Table 4.7).

Table 4.7 First scenario screen of BorrowBook

<table>
<thead>
<tr>
<th>Step</th>
<th>Type of Message</th>
<th>Sender</th>
<th>Message</th>
<th>Receiver</th>
<th>Widget</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Synchronous</td>
<td>Librarian</td>
<td>Enter-ID()</td>
<td>Library system</td>
<td>Text</td>
<td>Input</td>
</tr>
<tr>
<td>2</td>
<td>Synchronous</td>
<td>Library system</td>
<td>OK=Verify(ID)</td>
<td>Borrower</td>
<td>Button</td>
<td>Input</td>
</tr>
</tbody>
</table>

Also table 4.8 and 4.9 show another screens of BorrowBook sequence diagram.
When all of the screens are specified for a use case, developer can make a navigation map with the scenario screens extracted for each scenario of use case (Table 4.10).

Table 4.8 Second scenario screen of BorrowBook

<table>
<thead>
<tr>
<th>Step</th>
<th>Type of Message</th>
<th>Sender</th>
<th>Message</th>
<th>Receiver</th>
<th>Widget</th>
<th>Constrain</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Enter-ISBN()</td>
<td>Librarian</td>
<td>Text</td>
<td>Input</td>
</tr>
<tr>
<td>3.1</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Enter-CopyNo()</td>
<td>Librarian</td>
<td>Text</td>
<td>Input</td>
</tr>
<tr>
<td>3.2</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Create Record()</td>
<td>Loan</td>
<td>Button</td>
<td>Input</td>
</tr>
<tr>
<td>3.3</td>
<td>Synchronous</td>
<td>Library system</td>
<td>Show Record()</td>
<td>Loan</td>
<td>Table</td>
<td>Output</td>
</tr>
</tbody>
</table>

Table 4.9 Third Scenario screen of BorrowBook

<table>
<thead>
<tr>
<th>Step</th>
<th>Type of Message</th>
<th>Sender</th>
<th>Message</th>
<th>Receiver</th>
<th>Widget</th>
<th>Constrain</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Synchronous</td>
<td>Library System</td>
<td>Show message (Deny Request)</td>
<td>Library System</td>
<td>Label</td>
<td>Output</td>
</tr>
</tbody>
</table>

Table 4.10 Navigation map for BorrowBook use case

<table>
<thead>
<tr>
<th>Sequence of Screens</th>
<th>Scenarios of BorrowBook use case</th>
<th>01 Enter</th>
<th>02 Borrow</th>
<th>03 Deny Borrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
4.4.5. Mock-up Generating and Displaying in Library System

Now developer is ready to generate mock-ups in a mock-up builder environment based on scenario screens that are extracted in previous step. After mock-ups are generated, they will attach to scenario screens and then are displayed based on order in navigation map of each scenario. Figure 4.6 to 4.8 show mock-ups of BorrowBook use case.

Figure 4.6 First mock-up of BorrowBook use case

Figure 4.7 Second mock-up of BorrowBook use case
4.4.6 Generate User interface Requirements Specification in Library System

In final stage of this process, all of tables and diagrams in library project are shown in printable text format.

4.5. Main Functionality OF WEBSTUIRE

The list of core features of WEBSTUIRE based on the proposed approach includes:

Requirements Management, Actors Management, creating and managing use cases and scenarios, generating Traceability Matrix, creating and managing sequence diagram, generate scenario screen and navigation map, produce mock-ups, and generating the SRS which include user interface requirements. We explained about them in detail in next chapter.
CHAPTER 5-REQUIREMENTS ANALYSIS OF WEBSTUIRE

5.1. Introduction

This chapter takes a look at analysis of the WEBSTUIRE which is a scenario-based and prototype Support Tool for UI Requirements Elicitation in the UML software model. The requirements needed for WEBSTUIRE as well as functional and nonfunctional requirements will be analyzed. Analysis of the WEBSTUIRE is represented using use case diagram and class diagram.

5.2. System Overview in Detail

5.2.1. Functional Requirements Specification

Main functional requirements of WEBSTUIRE are outlined in Table 5.1:
Table 5.1 Functional Requirements of WEBSTUIRE

<table>
<thead>
<tr>
<th>R#</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>WEBSTUIRE shall allow the developer to create a project</td>
<td>1</td>
</tr>
<tr>
<td>R2</td>
<td>WEBSTUIRE shall allow the End-user to select/view a project</td>
<td>1</td>
</tr>
<tr>
<td>R3</td>
<td>WEBSTUIRE shall allow the developer to add requirements to a project</td>
<td>1</td>
</tr>
<tr>
<td>R4</td>
<td>WEBSTUIRE shall allow the developer to edit requirements in a project</td>
<td>1</td>
</tr>
<tr>
<td>R5</td>
<td>WEBSTUIRE shall allow the developer to delete requirements from a project</td>
<td>1</td>
</tr>
<tr>
<td>R6</td>
<td>WEBSTUIRE shall allow the End-user to view requirements of project</td>
<td>1</td>
</tr>
<tr>
<td>R7</td>
<td>WEBSTUIRE shall allow the developer to add actors to a project</td>
<td>1</td>
</tr>
<tr>
<td>R8</td>
<td>WEBSTUIRE shall allow the developer to edit actors in a project</td>
<td>1</td>
</tr>
<tr>
<td>R9</td>
<td>WEBSTUIRE shall allow the developer to delete actors from a project</td>
<td>1</td>
</tr>
<tr>
<td>R10</td>
<td>WEBSTUIRE shall allow the End-user to view actors of project</td>
<td>1</td>
</tr>
<tr>
<td>R11</td>
<td>WEBSTUIRE shall allow the developer to input individual events in a use case</td>
<td>1</td>
</tr>
<tr>
<td>R12</td>
<td>WEBSTUIRE shall allow the End-user to view events in a use case</td>
<td>1</td>
</tr>
<tr>
<td>R13</td>
<td>WEBSTUIRE shall allow the developer to manage events of use cases in a project by simple GUI interface</td>
<td>1</td>
</tr>
<tr>
<td>R14</td>
<td>WEBSTUIRE shall allow the developer to add use cases to a project</td>
<td>1</td>
</tr>
<tr>
<td>R15</td>
<td>WEBSTUIRE shall allow the developer to edit use cases in a project</td>
<td>1</td>
</tr>
<tr>
<td>R16</td>
<td>WEBSTUIRE shall allow the developer to delete use cases from a project</td>
<td>1</td>
</tr>
<tr>
<td>R17</td>
<td>WEBSTUIRE shall allow the End-user to view use case of project</td>
<td>1</td>
</tr>
<tr>
<td>R18</td>
<td>WEBSTUIRE shall allow the developer to associate actors with a use case</td>
<td>1</td>
</tr>
<tr>
<td>R19</td>
<td>WEBSTUIRE shall allow the developer to generate Scenarios base on selecting use case steps.</td>
<td>1</td>
</tr>
<tr>
<td>R20</td>
<td>WEBSTUIRE shall allow the End-user to view scenarios of project</td>
<td>1</td>
</tr>
<tr>
<td>R21</td>
<td>WEBSTUIRE shall allow the developer to generate a traceability matrix</td>
<td>1</td>
</tr>
<tr>
<td>R22</td>
<td>WEBSTUIRE shall allow the End-user to view scenarios of project</td>
<td>1</td>
</tr>
<tr>
<td>R23</td>
<td>WEBSTUIRE shall allow the developer to add sequence diagrams to a project</td>
<td>1</td>
</tr>
<tr>
<td>R24</td>
<td>WEBSTUIRE shall allow the developer to edit sequence diagrams in a project</td>
<td>1</td>
</tr>
<tr>
<td>R25</td>
<td>WEBSTUIRE shall allow the developer to delete sequence diagrams from a project</td>
<td>1</td>
</tr>
<tr>
<td>R26</td>
<td>WEBSTUIRE shall allow the developer to input messages of sequence diagram in a tabular format</td>
<td>1</td>
</tr>
<tr>
<td>R27</td>
<td>WEBSTUIRE shall allow the developer to enrich messages of sequence diagrams with UI’s widget</td>
<td>1</td>
</tr>
<tr>
<td>R28</td>
<td>WEBSTUIRE shall allow the developer to manage messages of sequence diagram in a project by simple GUI interface</td>
<td>1</td>
</tr>
<tr>
<td>R29</td>
<td>WEBSTUIRE shall allow the End-user to view sequence diagram of project</td>
<td>1</td>
</tr>
<tr>
<td>R30</td>
<td>WEBSTUIRE shall allow the developer to generate scenario screen from steps of sequence diagram.</td>
<td>1</td>
</tr>
<tr>
<td>R31</td>
<td>WEBSTUIRE shall allow the developer to produce mockups</td>
<td>1</td>
</tr>
<tr>
<td>R32</td>
<td>WEBSTUIRE shall allow the developer to generate a navigation map for scenario screens that are generated for each scenario.</td>
<td>1</td>
</tr>
<tr>
<td>R33</td>
<td>WEBSTUIRE shall allow the developer and End-user to show mockups base on navigation map</td>
<td>1</td>
</tr>
<tr>
<td>R34</td>
<td>WEBSTUIRE shall allow the End-user to give feedback about all of steps in process.</td>
<td>1</td>
</tr>
<tr>
<td>R35</td>
<td>WEBSTUIRE shall allow the developer to generate SRS document included UI requirements in text format</td>
<td>1</td>
</tr>
</tbody>
</table>

5.2.2 Non-functional Requirements Specification

Main non-functional requirements of WEBSTUIRE are given Table 5.2:
Table 5.2 Non-functional Requirements of WEBSTUIRE

<table>
<thead>
<tr>
<th>R#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R36</td>
<td>WEBSTUIRE shall be used easily by End-users and developers (usability).</td>
</tr>
<tr>
<td>R37</td>
<td>WEBSTUIRE shall be learned in ten minutes by the developers (usability).</td>
</tr>
<tr>
<td>R38</td>
<td>WEBSTUIRE shall provide meaningful and instructive error message for users help to increase usability of the system (usability).</td>
</tr>
<tr>
<td>R39</td>
<td>WEBSTUIRE shall be easily expanding for future purpose with the components which could be easily modified in future (scalability).</td>
</tr>
<tr>
<td>R40</td>
<td>WEBSTUIRE shall design the structure which can be easily maintained and expanded with the least modification of components and data structure (maintainability).</td>
</tr>
<tr>
<td>R41</td>
<td>WEBSTUIRE shall be accessible through multiple web browser platforms (portability).</td>
</tr>
<tr>
<td>R42</td>
<td>WEBSTUIRE shall provide secure access to the database (security).</td>
</tr>
<tr>
<td>R43</td>
<td>WEBSTUIRE should be able examine the permission of each authorized user for accessing different services within the system (security).</td>
</tr>
<tr>
<td>R44</td>
<td>WEBSTUIRE shall be written in C#</td>
</tr>
<tr>
<td>R45</td>
<td>WEBSTUIRE shall store Microsoft SQL server 2000 as optional data storage mechanism.</td>
</tr>
<tr>
<td>R46</td>
<td>WEBSTUIRE shall develop a windows based tools</td>
</tr>
</tbody>
</table>

5.3. Use Case Diagram

Figure 5.3 shows the use case diagram of WEBSTUIRE. There are two types of actor that can interact with WEBSTUIRE as Developer and End-user. In this diagram, various use cases of WEBSTUIRE are presented functionalities with which each actor can interact.
5.3.1. Use Case Model

Figure 5.1 Use Case diagram of WEBSTUIRE

5.3.2. WEBSTUIRE Use Cases Description

In this section each use case in the use case diagram is explained in the use case description as detailed. Use case description makes clears the preconditions, post conditions, actions, alternative and Scenarios for each use case. Table 5.3 till Table 5.17 show the use case descriptions for the WEBSTUIRE.
<table>
<thead>
<tr>
<th><strong>Use Case ID:</strong> USE CASE-1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use Case Name:</strong> Manage Requirement</td>
</tr>
<tr>
<td><strong>Actors:</strong> Developer</td>
</tr>
<tr>
<td><strong>Brief Description:</strong> provide managing of requirements</td>
</tr>
<tr>
<td><strong>Preconditions:</strong> The user has clicked on the link “Requirements”</td>
</tr>
<tr>
<td><strong>Action Description:</strong></td>
</tr>
</tbody>
</table>
| 1. If Developer chooses to add a requirement  
  a. Include AddRequirement |
| 2. If Developer chooses to edit a requirement  
  a. Include EditRequirement |
| 3. If Developer chooses to delete a requirement  
  a. Include DeleteRequirement |
| **Alternative Flow 1:** |
| 1. At any time the Developer may leave the Requirements tab |
## UC2 – Add Requirement

Table 5.4 WEBSTUIRE Use Case for Add Requirement

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Add Requirement</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>add new requirement to a project</td>
</tr>
<tr>
<td>Pre Conditions:</td>
<td>Developer selects to add a requirement</td>
</tr>
</tbody>
</table>

**Action Description:**

1. Developer selects type of requirement whether it is functional or non-functional
2. Developer selects priority of the requirement
3. Developer enters the description of requirement in textbox
4. Developer clicks “save”
5. If textbox is empty
   a. Prompt User to enter a requirement in the textbox

Repeat steps 1-5 until textbox is not empty
6. System adds requirement to the list and system displays “You successfully add requirement!”

**Post Conditions:** A requirement is added to the database

**Alternative Flow 1:**

1. At any time the Developer may leave the Requirements tab

**Scenarios:**

- SuccessfulRequirementAdding
- EmptyRequirementName

---

58
### UC3 – Edit Requirement

Table 5.5 WEBSTUIRE Use Case for Edit Requirement.

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Edit Requirement</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>provide editing of requirements</td>
</tr>
<tr>
<td>Pre Conditions:</td>
<td>User selects to edit a requirement</td>
</tr>
</tbody>
</table>

**Action Description:**
1. Developer selects a requirement from the existing requirements list
2. Developer edits requirement text, priority, and/or the type of requirement
3. User clicks “save changes”
4. If textbox is empty
   a. Prompt developer to enter the requirement in the box
   Repeat steps 1-4 until textbox is not empty
5. System edits requirement in list and system displays “You successfully edit requirement!”

**Post Conditions:** A requirement is modified and updated.

**Scenarios:**
- SuccessfulRequirementEditing
- EmptyRequirementName
### UC4 – Delete Requirement

Table 5.6 WEBSTUIRE Use Case for Delete Requirement.

<table>
<thead>
<tr>
<th>Use Case ID: USE CASE-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name: Delete Requirement</td>
</tr>
<tr>
<td>Actors: Developer</td>
</tr>
<tr>
<td>Brief Description: delete a requirement from list</td>
</tr>
<tr>
<td>Preconditions: User selects to delete a requirement</td>
</tr>
</tbody>
</table>

**Action Description:**
1. Developer selects a requirement from the list
2. Developer clicks “Delete Requirement”
3. System delete requirement from list and display message “You successfully delete Requirements!”

**Post Conditions:** A requirement is deleted.

**Scenarios:**
- SuccessfulRequirementDeleting

### UC5 – Use Case Management

Table 5.7 WEBSTUIRE Use Case Description for Manage Use Cases

<table>
<thead>
<tr>
<th>Use Case ID: USE CASE-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name: Manage Use Cases</td>
</tr>
<tr>
<td>Actors: Developer</td>
</tr>
<tr>
<td>Brief Description: provide managing of Use Case</td>
</tr>
<tr>
<td>Preconditions: The user has clicked on the link “Use Cases”.</td>
</tr>
</tbody>
</table>

**Action description:**
1. If Developer chooses to Create a use case
   a. Include CreateUseCase
2. If Developer chooses to edit a use case
   a. Include EditUseCase
3. If Developer selects to delete a use case
   a. Include DeleteUseCase

**Alternative Flow 1:**
1. At any time the Developer may cancel the operation
UC6 – Create Use Case

Table 5.8 WEBSTUIRE Use Case for Create Use Case

<table>
<thead>
<tr>
<th>Use Case ID: USE CASE-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name: Create Use Cases</td>
</tr>
<tr>
<td>Actors: Developer</td>
</tr>
<tr>
<td>Brief Description: add a new Use Case to list</td>
</tr>
<tr>
<td>Pre Conditions: Developer chooses to create a use case</td>
</tr>
</tbody>
</table>

**Action description:**

1. Developer clicks on Create New button and inputs the name of the use case.
2. Developer enters use case pre conditions, if any.
3. For each step in the use case:
   a. Developer selects type of the step which includes: if, while, basic etc.
   b. System auto generates the step Index using the built-in branching algorithm.
   c. Developer enters Action description.
   d. Developer clicks Submit Step button.
4. Developer enters use case post conditions, if any, and clicks Submit Use Case.
5. If use case name box is empty:
   a. system displays message “use case name is empty”
Repeat step 1-5 until Developer inputs a use case name.
7. If System finds a duplicate use case name:
   a. system displays message “Input a unique use case name”.
Repeat steps 6-7 until Developer inputs a unique use case name.
8. System adds use case to list

**Post Conditions:** If completed, use case is added to the database.

**Alternative Flow 1:**

1. At any time the Developer may leave the Use Case tab.

**Scenarios:**

SuccessfulUseCaseCreation
EmptyUseCaseName
**UC7 – Edit Use Case**

Table 5.9 WEBSTUIRE Use Case for Edit Use Case

<table>
<thead>
<tr>
<th>Use Case ID: USE CASE-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name: Edit Use Cases</td>
</tr>
<tr>
<td>Actors: Developer</td>
</tr>
<tr>
<td>Brief Description: provide editing of Use Cases</td>
</tr>
<tr>
<td>Preconditions: Developer chooses an existing use case for editing.</td>
</tr>
</tbody>
</table>

**Action description:**

1. System displays list of existing Use Case
2. Developer selects a use case from the list
3. System retrieves all Use Case Steps and displays them.
4. Developer edits desired fields in the use case form.
5. If Developer wants to add more steps to the use case:
   a. Developer selects type of the step (such as if, while, basic etc.)
   b. System auto generates the step Index using the built-in branching algorithm.
   c. Developer enters Action description.
   d. Developer clicks Submit Use Case button.
6. Developer clicks submit
7. If use case name is empty:
   a. system displays message “use case name is empty”.
Repeat steps 1-7 until Developer inputs a use case name
8. System searches for duplicate use case names.
9. If System finds duplicate use case names:
   a. system displays message “Input a unique use case name”.
Repeat steps 7-8 until Developer inputs unique use case name.

**Post Conditions:** A use case is modified and updated in database.

**Alternative Flow 1:**
1. At any time the Developer may leave the Scenario tab.

**Scenarios:**

SuccessfulUseCaseEditing
EmptyUseCaseName
UC8 – Delete Use Case

Table 5.10 WEBSTUIRE Use Case for Delete Use Case

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Delete Use Case</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>delete a use case from list</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>User selects to delete a Use Case</td>
</tr>
</tbody>
</table>
| Action Description: | 1. Developer selects a use case from the list  
2. Developer clicks “Delete Use Case”  
3. System delete use case in list and display message “You successfully delete Use Case!” |
| Post Conditions: | A Use Case is deleted from database. |
**UC9 Generate Scenario**

Table 5.11 WEBSTUIRE Use Case for Generate Scenario

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Generate Scenario</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>add a scenario to database</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>Developer opens a system and clicks on the Scenarios tab.</td>
</tr>
</tbody>
</table>

**Action Description:**

1. Developer selects one use case from the list box ‘Select Use Case’
2. The system loads all events of the use case for selection.
3. Developer selects one or more events that correspond to the scenario that is being described.
4. Developer clicks on the generate button.
5. System automatically generates scenario in the scenario grid.
6. Developer enters scenario name and clicks on the save button.
7. If scenario name is empty:
   a. system displays message “scenario name is empty”.
   Repeat steps 6-7 until Developer inputs a scenario name
8. System searches for duplicate scenario names.
9. If System finds duplicate scenario names:
   a. system displays message “Input a unique scenario name”.
   Repeat steps 7-9 until Developer inputs unique scenario name.
10. The scenario is saved to the database.

**Post Conditions:** A scenario description is displayed to the developer and saved in database.

**Scenarios:**

- Successful Scenario Generation
- Scenario Navigate Out
- Duplicate Scenario Found
UC10 – Sequence Diagram Management

Table 5.12 WEBSTUIRE Use Case Description for Manage Sequence Diagram

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Manage Sequence Diagram</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>provide managing of sequence diagram</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>The developer has clicked on the link “Sequence Diagram”.</td>
</tr>
<tr>
<td>Action description:</td>
<td>1. If Developer chooses to Create a Sequence Diagram</td>
</tr>
<tr>
<td></td>
<td>a. Include Create Sequence Diagram</td>
</tr>
<tr>
<td></td>
<td>2. If Developer chooses to edit a Sequence Diagram</td>
</tr>
<tr>
<td></td>
<td>a. Include Edit Sequence Diagram</td>
</tr>
<tr>
<td></td>
<td>3. If Developer selects to delete a Sequence Diagram</td>
</tr>
<tr>
<td></td>
<td>a. Include Delete Sequence Diagram</td>
</tr>
<tr>
<td>Alternative Flow 1:</td>
<td>1. At any time the Developer may cancel the operation</td>
</tr>
</tbody>
</table>
## UC11 – Create Sequence Diagram

Table 5.13 WEBSTUIRE Use Case for Create Sequence Diagram

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Create Sequence Diagrams</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>add a new sequence diagram to database</td>
</tr>
<tr>
<td>Pre Conditions:</td>
<td>User chooses to create a sequence diagram</td>
</tr>
</tbody>
</table>

### Action description:
1. Developer clicks on Create New button and inputs the name of the Sequence Diagram.
2. Developer enters life lines.
3. For each step in the sequence diagram:
   a. Developer selects type of the step which includes: Synchronous, Asynchronous, Reply.
   b. System auto generates the step Index using the built-in branching algorithm.
   c. Developer selects sender of message.
   d. Developer enters message
   e. Developer selects receiver of message
   f. Developer selects constraint of message (such as input, output)
   g. Developer selects type of widget of UI based on widget rules.
   h. Developer clicks Submit Step button.
4. Developer clicks Submit Sequence Diagram.
5. If sequence diagram name box is empty:
   a. System displays message “sequence diagram name is empty”
Repeat step 1-5 until Developer inputs a sequence diagram name.
7. If System finds a duplicate sequence name:
   a. System displays message “Input a unique sequence name”.
Repeat steps 6-7 until Developer inputs a unique sequence name.
8. System adds sequence diagram to list
**Post Conditions:** If completed, sequence diagram is added to the database.

**Alternative Flow 1:**

1. At any time the Developer may leave the Sequence tab.

**Scenarios:**

- Successful Sequence Creation
- Empty Sequence Name

### UC12 – Edit Sequence

Table 5.14 WEBSTUIRE Use Case for Edit Sequence

<table>
<thead>
<tr>
<th><strong>Use Case ID:</strong></th>
<th>USE CASE-12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use Case Name:</strong></td>
<td>Edit Sequence diagram</td>
</tr>
<tr>
<td><strong>Actors:</strong></td>
<td>Developer</td>
</tr>
<tr>
<td><strong>Brief Description:</strong></td>
<td>provide editing of sequence diagram</td>
</tr>
<tr>
<td><strong>Pre Conditions:</strong></td>
<td>Developer chooses an existing Sequence diagram for editing.</td>
</tr>
</tbody>
</table>

**Action description:**

1. System displays list of existing sequence diagram
2. Developer selects a sequence diagram from the list
3. System retrieves all sequence diagram Steps and displays them.
4. Developer edits desired fields in the sequence diagram form.
5. If Developer wants to add more steps to the sequence:
   a. Developer selects type of the step which includes: Synchronous, Asynchronous, Reply.
   b. System auto generates the step Index using the built-in branching algorithm.
   c. Developer selects sender of message.
   d. Developer enters message
   e. Developer selects receiver of message
   f. Developer selects constraint of message(such as input, output)
   g. Developer selects type of widget of UI base on widget rules.
   h. Developer Submit Step button.
6. Developer clicks submit sequence
7. If sequence name is empty:
   a. system displays message “sequence name is empty”.

Repeat steps 1-7 until Developer inputs a sequence name
8. System searches for duplicate sequence names.
9. If System finds duplicate sequence names:
   a. system displays message “Input a unique sequence name”.

Repeat steps 7-8 until Developer inputs unique sequence name.
9. System saves sequence.

Post Conditions: A sequence is modified and updated in database.

Alternative Flow 1:
1. At any time the Developer may leave the sequence tab.

Scenarios:
- Successful Sequence Editing
- Empty Sequence Name

### UC13 – Delete Sequence

Table 5.15 WEBSTUIRE use case for Delete Sequence.

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Delete Sequence</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>delete a sequence from list</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>developer selects to delete a Sequence</td>
</tr>
<tr>
<td>Action Description:</td>
<td></td>
</tr>
<tr>
<td>1. Developer selects a sequence from the list</td>
<td></td>
</tr>
<tr>
<td>2. Developer clicks “Delete Sequence ”</td>
<td></td>
</tr>
<tr>
<td>3. System delete sequence from list and display message “You successfully delete Sequence!”</td>
<td></td>
</tr>
<tr>
<td>Post Conditions:</td>
<td>A Sequence is deleted from database.</td>
</tr>
</tbody>
</table>
**UC14 Generate Scenario Screen**

Table 5.16 WEBSTUIRE Use Case for Generate Scenario Screen

<table>
<thead>
<tr>
<th>Use Case ID:</th>
<th>USE CASE-14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name:</td>
<td>Generate Scenario Screen</td>
</tr>
<tr>
<td>Actors:</td>
<td>Developer</td>
</tr>
<tr>
<td>Brief Description:</td>
<td>determine scenario screen</td>
</tr>
</tbody>
</table>

**Pre Conditions:** Developer opens a system and clicks on the Scenario Screen tab.

**Action Description:**

1. Developer selects one Sequence from the list box ‘Select Sequence’
   2. The system loads table of the sequence for selection.
   3. Developer selects one or more steps of message that correspond to the screen of scenario.
   4. Developer clicks on the generate button.
   5. System automatically generates screen in the screen grid.
   6. Developer enters screen name and clicks on the save button.
   7. If screen name is empty:
      a. system displays message “screen name is empty”.
     Repeat steps 6-7 until Developer inputs a screen name
   8. System searches for duplicate screen names.
   9. If System finds duplicate screen names:
      a. system displays message “Input a unique screen name”.
     Repeat steps 7-9 until Developer inputs unique screen name.

10. The screen is saved to the database.

**Post Conditions:** A screen description is displayed to the developer and saved in database.

**Scenarios:**

- SuccessfulScreenGeneration
- DuplicateScreenFound
UC15 Generate Navigation Map

Table 5.17 WEBSTUIRE Use Case for Generate Navigation Map.

<table>
<thead>
<tr>
<th>Use Case ID: USE CASE-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Name: Generate Navigation Map</td>
</tr>
<tr>
<td>Actors: Developer</td>
</tr>
<tr>
<td>Brief Description: determine navigation map table</td>
</tr>
<tr>
<td>Preconditions: Developer opens a system and clicks on the tab of Navigation Map.</td>
</tr>
</tbody>
</table>

**Action Description:**

1. Developer selects one Use Case from the list box ‘Select Use Case’
2. The system loads related Scenarios and Scenario Screens.
3. Developer selects one scenario from Available Scenarios
4. Developer transfer one or more screens from Scenario Screen Available to Associate Scenario Screen.
5. System shows in the navigation grid.
6. Developer clicks on the Save button.
7. The navigation map is saved to the database.

**Post Conditions:** A navigation Map is displayed to the developer and saved in database.

5.4. Object Model

The goal of the analysis phase is to identify the main artifacts, system’s entities and visualization of main features of the system using various types of diagrams, graphs and models such as use case model, object model and so forth.

Objects are structures of the artifacts through the system identified by the object model. Figure 5.2 represents the class diagram of WEBSTUIRE which visualizes the name and attributes of each object and the relationship between them.

There is a data object class that keeps information of the Project. It can have collection of UseCases, Actors, Requirements, sequences, User, and comments.
The UseCase can have a collection of UseCaseEvents and depend on the Actor, Requirement, and Sequence object classes. The UseCaseEvents can have a collection of Scenarios. The Sequence data object class can have a collection of SequenceSteps. Each SequenceStep can have a collection of LifeLines and widgets. Each ScenarioScreen can have a collection of SequenceSteps and depend on the Scenario, NavigationMap, and Mockup.

Figure 5.2 Object Model Of WEBSTUIRE
CHAPTER 6-DESIGN AND IMPLEMENTATION

6.1. Introduction

This chapter describes details of design and implementation of WEBSTUIRE. In the previous chapter, we elicited and analyzed all the requirements required for developing the WEBSTUIRE tool. In this chapter based on gathered information, the detailed design of WEBSTUIRE is represented by using different object oriented modeling diagrams. Also, architecture and components of WEBSTUIRE are explained to clarify implementation of the tool. Most important screen shots of WEBSTUIRE are shown to display design of the user interface of the tool. Finally, software and hardware requirements needed for development of WEBSTUIRE are specified in the table.

6.2. Constructing WEBSTUIRE Algorithm

This section discusses the construction rules which affect WEBSTUIRE. There is a particular algorithm behind the WEBSTUIRE that enables it to analyze user requirements in order to propose user interface SRS within an examined application system. This specific algorithm is designed as the following:

By using WEBSTUIRE, software developers can elicit user requirements from the users of a particular software system. Also, WEBSTUIRE provides helpful facility for software developers to create use cases and scenario of main functionality of the examined software system. By using use cases and requirements, WEBSTUIRE will be able to analyze user requirements and then it starts to create traceability matrix to make sure about users’ requirements. Based on analysis on requirements, use cases, scenarios, and traceability matrix and based on feedbacks gotten from users, WEBSTUIRE provides helpful ability to create sequence diagram that enriched with UI widgets in order to generate scenario screens that made mock-ups of the proposed
system. Likewise, software developers can create navigation map for each scenario. Finally WEBSTUIRE generates UI-SRS based on information gathered from examined software system.

6.3. WEBSTUIRE Design

6.3.1. Package Diagram

Package diagrams enable developers to organize model elements into groups, making it simpler and easier to understand other UML diagrams. Packages are depicted as use-case diagrams and class diagrams because these models have a tendency to grow.

WEBSTUIRE contains the following packages:

- Database package consist of classes for connecting to the database of system.
- Database Services package consist of classes for calling database services
- ASP.net Classes package consist of ASP.net classes of the WEBSTUIRE tool explained by object model section in last chapter.
- ASPX files package includes ASPX files which construct the WEBSTUIRE interface.

Figure 6.1 visualizes the package diagram of WEBSTUIRE.
6.3.2. Class Diagram

Class diagram depicts objects of the system, their interrelationships, the operation and attributes of the classes. Class diagram of WEBSTUIRE indicates the name of the class, the attributes of the class and the list of the methods inside each object. Figure 6.2 depicts the package diagram of WEBSTUIRE in detail and illustrates the list of classes inside each package. In previous chapter, the list of attributes of each class is identified by object model.
6.3.3 Entity Relationship Diagram (ERD)

Entity relationship diagram depicts the interrelationships between entities in a database. Figure 6.3 represents entities and database tables of the WEBSTUIRE with relationships among them using ERD.
6.4. Architectural Design of WEBSTUIRE

Architectural design of a system portrays the structure of the physical components inside the system and relationship between them. MVC pattern is essentially an architectural model of sorting the user experience for visualizing design of application into three components: the Model, the View, and the Controller. The MVC model separates the components of a Web application usually used for modeling architecture of web based applications since it contains especially view of the system. The general data flow of architectural model is diagrammed in Figure 6.4.
WEBSTUIRE applies MVC architecture as the design pattern. To clarify, the detail of how the application is segmented some architectural elements of WEBSTUIRE are shown below:

- **Controller**,

  Controller is a file that accept request from users and interpret it and command the Model or the View to change as appropriate. It actually is the bridge between the Model and the View and it contains class files of WEBSTUIRE written in C# language.

- **Model**

  ADO.net is the business logic of application that modeled the objects inside WEBSTUIRE. It provides consistent access to the database and can save, manipulate, and retrieve object models of the system.

- **View**

  View can consist of every type of interface given to the user. In ASP.NET the view is the set of web pages presented by a web application. ASPX is an Active Server Page Extended. This webpage includes following information: ASPX description, user guides and software to open/edit ASPX format file. ASP.net constructs the user interface of the WEBSTUIRE in files with .aspx format which construct the view of the system and return the response to the users.
- **Browser**

Any web browser can be used for WEBSTUIRE such as IE or Firefox. Users can access WEBSTUIRE by using any version of web browsers.

- **Database**

Database stores all the data of WEBSTUIRE. Database type of WEBSTUIRE is SQL 2000.

### 6.5. User Interface Design

In this section, the screen shots of the interface of WEBSTUIRE are displayed. The following figures show how WEBSTUIRE performs desired tasks through its GUI (Graphical User Interface).

In the login page, both users and developers can login to the system. WEBSTUIRE users start with this page and enter the username and password. The Log in and Sign up buttons allow users to log in to the system when necessary.

![Figure 6.5 Log In page](image.png)
Create account page provides registration to the system for users as developer or end-user.

Figure 6.6 Sign up page
Developers can create new project by entering the title and information of the system and press the save button. The Edit and Delete buttons allow users the ability to modify and delete their input when necessary.

![Create Project page](image)

Figure 6.7 Create Project page
Developers can manage all the requirements of the defined system using the following page. The user can create a new requirement by choosing the type of requirements and its priority, and by entering the description of requirement. The Edit and Delete buttons allow the users to edit contents or to remove it by pressing Delete button. Each requirement is classified as functional or non-functional. WEBSTUIRE provides its users the capability to prioritize the requirements on a scale of 1 to 3, with 1 being the highest priority.

Figure 6.8 Requirements Management page
Developers can manage actors that interact with the defined system using the following page.

![Actor Management page](image)

Figure 6.9 Actor Management page

The Actors list shows existing actors. Adding a new actor to the system is provided by entering the Actor Name and then pressing Add button. Delete removes the name of a selected actor from the list. When an actor is deleted from list, use cases mappings associated with it are cleared but WEBSTUIRE does not delete an actual use case from the project.
Developers can manage the use cases and steps of the use cases that relate with the defined system using the following page.

WEBSTUIRE establishes a simple way of creating and editing use cases. WEBSTUIRE allows the user to add event step of use case easily. Figure 6.10 shows

![Use Case Management page](image)

Figure 6.10 Use Case Management page
the snapshot of use case definition in progress. User presses Add New button and enters use case name, use case description and pre condition. Specifying use case steps is the next step. Use Case steps are numbered and each step expresses the kind of logical operation it belongs to, such as basic, if, else, while, etc. Numbering for each step is done automatically. The user just needs to select the type of step, enters action of step, and click Add Step button. This is a simple operation but it becomes more complicated if the user selects “If” from the Type drop down list, step 2.1 is automatically generated by WEBSTUIRE. The user then enters the condition in description and clicks Add Step then this step goes into the grid. From now on WEBSTUIRE would automatically numbers sub-steps continuing with 2.2, 2.3 etc. If the user clicks on “Sub Step” button, he or she shows indicating continuation of the current conditional decision. When the user selects Continue Type, it indicates that the previous conditional statement block becomes continues. While the users finished this construct all the user needs to click on “End Sub”. The “Last Step” Button returns the number of last step when the user clicks on this button.

Other buttons provide relatively simple operations. Edit button updates the use case from the existing use cases. Delete button removes the currently selected use case from the list and clears the contents of the Use Case Details area.
Developers can associate the use cases to actors defined for system using the following page.

![WEB STUIRE](image)

**Figure 6.11 Associated Use case to Actor page**

Looking at this form, it is easy to get an entire perspective of the system by understanding the use cases associated with an actor and considering inheritance relationships among the actors. Figure 6.11 shows the Associate Use Cases-Actors process in action. This form acts very intuitively as it appears just by taking a glance at the UI. Actor inheritance relationships or use case associations are set by first selecting an actor. Other lists then populate only the actors that can inherit from the selected actor or the use cases that can be associated with that actor. The user selects an actor from the “Available Actors (to inherit)” list and moves it to the “Inherited Actors” list to create this inheritance. If an inherited actor is no longer necessary, it can be cancelled.
by moving it back to the left hand side list. The >> and << buttons move the actors and create/cancel the relationships. Similarly, all use cases that can be associated with an actor are listed in “Available Use Cases” list for currently selected actor from the main “Select An Actor” list. The >> and << Buttons provide the functionality of associating and removing the association just like the actor inheritance functionality described above. Option “Use Case without Actor” shows the list of use cases that haven’t been associated with an actor.
Developers can associate the use cases to requirements defined for the system and create the traceability matrix using the following page.

![Traceability Matrix page](image)

Figure 6.12 Traceability Matrix page

Figure 6.12 shows the Traceability Matrix page that lets the user select a use case and then select one or more requirements from available requirements to associate with that use case. Again the association is done using the >> button. The Select Use Case combo box allows picking a use case and the Available Requirements list box shows all of the requirements and their texts that are available for association. The requirements association works just like the associate actors functionality described before. While these associations are being made, WEBSTUIRE simultaneously creates the traceability matrix in real time. It keeps changing as these associations are taking place. If a requirement is deselected to remove the association, the traceability matrix is
updated to unmark that mapping. When the user moves on to a different use case, it stores the associations of the current use case and continues to add to the traceability matrix for the new use case. This frees the user from remembering previous use cases associations with requirements as the matrix readily shows all use case-requirement associations. If the user goes back to a use case, the associated and available requirements are reloaded from the point they were left off. This creates a simple mechanism that allows the user to select use cases and just focus on associating desired requirements, leaving the rest to WEBSTUIRE, which builds the traceability matrix behind the scene and shows it on screen.
Developers can generate scenarios from steps of use cases defined for the system using the following page.

![Figure 6.13 Scenario Generation Page](image)

The UI screenshot shown in Figure 6.13 is taken during the generation of a scenario using WEBSTUIRE. The approach is very similar to the automatic generation we have seen so far in use case specification, traceability matrix definition, etc. The user selects the use case to create a scenario. WEBSTUIRE inhabits all the individual steps from that use case in the Select Steps list box. The user selects all relevant events that create the scenario that he or she is interested in and clicks on the Generate button, that’s basically all what the user has to do. WEBSTUIRE automatically generates the grid view of scenario events. The user just needs to name the scenario and save it to associate the scenario with the use case. All remaining controls in the Scenarios tab are
intuitive and work as one would expect. The user can select an existing scenario to review it and delete it if desired.
Developers can manage the sequence diagrams that relate with use case of the defined system using the following page.

Figure 6.14 Sequence Management page
WEBSTUIRE establishes a simple way of creating and editing sequence diagram. It allows the user to easily add event step of sequence diagram that enriched with user interface widget. Figure 6.14 shows the snapshot of sequence diagram definition in progress. The user selects the use case to create a sequence diagram. The user presses Add button and enters sequence name. Specifying sequence steps is next step. Sequence steps are numbered and each step expresses type of message such as synchronous, asynchronous or reply. The numbering for each step is similar to automatic generation we have seen so far in use case specification. The user just needs to select type of message, sender of message, and receiver of message. Then, the user must enter message, select constraint of message and widget, and finally click Add Step button. The sub step number such as 3.1 is automatically generated by WEBSTUIRE when the user clicks on “Sub Step” button then this step goes into the grid. From now on WEBSTUIRE would automatically number sub-steps continuing with 3.2, 3.3 etc. When the user finished this construct all the user needs is to click on “End Sub”. The “Last Step” button returns the number of last step when the user clicks on this button. The user can add new lifeline and widget that he or she needs to the drop down list by clicking on the ”Add Lifeline” and “Add widget” buttons.

Other buttons provide relatively simple operations. Edit button updates the sequence from the existing sequences. Delete buttons removes the currently selected sequence from the list and clears the contents of the Sequence Details area.
Developers can generate the scenarios screen from steps of sequence diagrams defined for system using the following page.

![WEBSTUIRE Scenario Screen Generation page](image)

**Figure 6.15 Scenario Screen Generation page**

The UI screenshot shown in Figure 6.15 is taken during the generation of a scenario screen using WEBSTUIRE. The approach is very similar to the automatic generation we have seen so far in scenario generation. The user selects the use case and the sequence to create a scenario screen. WEBSTUIRE populates all the individual steps from that sequence in the Select Steps list box. The user selects all relevant events that create the scenario screen that he or she is interested in and clicks on the Generate button. That’s basically all what the user has to do. WEBSTUIRE automatically generates the grid view of scenario events. The user just needs to name the scenario.
screen and save it to associate the scenario screen with the sequence. All remaining controls in the Scenarios Screen page are intuitive and work as one would expect. The user can select an existing scenario screen to review it and delete it if desired.
Developers can generate the navigation map for all of scenarios that are related with sequence diagrams based on generated scenario screens using the following page.

Figure 6.16 Navigation Map Generation page
Developers can attach the mock-ups to scenario screens using the following page. Mock-ups are generated in the mock-up builder environment based on scenario screen. And then developer can upload them on related scenario screens. Finally they are displayed based on the navigation map.

![Mockup Page]

Figure 6.17 Upload Mockup Page
Figure 6.18 Show Mock-up Page
Developers can generate the SRS included in user interface requirements in different format such as PDF, MS Word using the following pages.

The users who log in to WEBSTUIRE can just view all of the stages of the selected system.
### SRS Report

**System Info:**
Library system is an interactive system aimed to keep and update all information in library.

#### Requirement

<table>
<thead>
<tr>
<th>Requirement Name</th>
<th>Type</th>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Functional</td>
<td>1</td>
<td>The library system shall allow librarians and borrowers to borrow books.</td>
</tr>
<tr>
<td>R2</td>
<td>Functional</td>
<td>1</td>
<td>The library system shall allow librarians and borrowers to return books.</td>
</tr>
<tr>
<td>R3</td>
<td>Functional</td>
<td>1</td>
<td>The library system shall allow borrowers to borrow a copy of a book.</td>
</tr>
<tr>
<td>R4</td>
<td>Functional</td>
<td>1</td>
<td>The library system shall allow users to update the status of books.</td>
</tr>
<tr>
<td>R5</td>
<td>Functional</td>
<td>1</td>
<td>The library system shall allow users to view the status of books.</td>
</tr>
<tr>
<td>R6</td>
<td>Functional</td>
<td>1</td>
<td>The library system shall allow borrowers to return borrowed books.</td>
</tr>
<tr>
<td>R7</td>
<td>Functional</td>
<td>1</td>
<td>The library system shall allow librarians to extend due dates for books.</td>
</tr>
</tbody>
</table>

#### Actors And Use Case Relation

<table>
<thead>
<tr>
<th>Actor Name</th>
<th>UICName</th>
</tr>
</thead>
<tbody>
<tr>
<td>librarian</td>
<td>CheckBookStatus</td>
</tr>
<tr>
<td>librarian</td>
<td>ReturnBook</td>
</tr>
<tr>
<td>librarian</td>
<td>UnBorrowedBook</td>
</tr>
<tr>
<td>librarian</td>
<td>ExtendDate</td>
</tr>
<tr>
<td>borrower</td>
<td>BorrowBook</td>
</tr>
<tr>
<td>borrower</td>
<td>LoginSystem</td>
</tr>
<tr>
<td>borrower</td>
<td>CheckBookStatus</td>
</tr>
<tr>
<td>borrower</td>
<td>ReturnBook</td>
</tr>
<tr>
<td>borrower</td>
<td>ExtendDate</td>
</tr>
<tr>
<td>librarian</td>
<td>BorrowBook</td>
</tr>
<tr>
<td>librarian</td>
<td>LoginSystem</td>
</tr>
<tr>
<td>librarian</td>
<td>CheckBook</td>
</tr>
</tbody>
</table>

#### Use Case And Steps

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Pre Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BorrowBook</td>
<td>Librarian must log in to system</td>
<td>borrower borrows book</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps</th>
<th>Type</th>
<th>Number</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>1</td>
<td></td>
<td>Librarian enters membership card validationSuccessful</td>
</tr>
<tr>
<td>Basic</td>
<td>2</td>
<td>1</td>
<td>Librarian enters title and copies of book</td>
</tr>
<tr>
<td>Basic</td>
<td>3</td>
<td></td>
<td>System updates circulation records and keeps due date</td>
</tr>
<tr>
<td>Obstruct</td>
<td>4</td>
<td></td>
<td>show message 'File Request'</td>
</tr>
<tr>
<td>Basic</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Use Case And Requirement Relation

<table>
<thead>
<tr>
<th>RequirementName</th>
<th>UICName</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>LoginSystem</td>
</tr>
<tr>
<td>R2</td>
<td>CheckBookStatus</td>
</tr>
<tr>
<td>R3</td>
<td>SearchBook</td>
</tr>
<tr>
<td>R4</td>
<td>ReturnBook</td>
</tr>
<tr>
<td>R5</td>
<td>UnBorrowedBook</td>
</tr>
<tr>
<td>R6</td>
<td>ExtendDate</td>
</tr>
<tr>
<td>R7</td>
<td>BorrowBook</td>
</tr>
</tbody>
</table>

#### Scenario's

| Scenario's |
|------------|-----------|
| 1          | Register User |
Figure 6.20 continue of SRS Report page
Figure 6.21 Comment Report page

<table>
<thead>
<tr>
<th>Reader</th>
<th>Message</th>
<th>Send Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>azza-admin</td>
<td>library system facilitates the interaction with system</td>
<td>2010/05/01 12:27:27</td>
</tr>
<tr>
<td>azza-admin</td>
<td>FC3 must also search based on the combination of the field</td>
<td>2010/05/02 02:45:55</td>
</tr>
<tr>
<td>azza-admin</td>
<td>an administrator also needs to add</td>
<td>2010/05/02 02:45:49</td>
</tr>
</tbody>
</table>
The users who log in to WEBSTUIRE can just view all of the stages of the selected system. But in all of the pages the users and developers who log in to system can leave their comments about each stage. Finally, developers can get a report of the user’s comments and this can help them to refine the system based on user’s comments and produce a complete SRS.
6.6 Tool Implementation

6.6.1 Development of WEBSTUIRE

WEBSTUIRE is a web based application which it runs on WWW protocol. There are several applications that have been used for development of web-based systems like WEBSTUIRE. In the following, some important components needed for WEBSTUIRE development are described.

6.6.1.1 Database Server

Database server is an important application which is dedicated to database storage and retrieval that belongs to the WEBSTUIRE. Microsoft SQL Server 2000 is the database server for developing WEBSTUIRE.

6.6.1.2 Web Server

Web server is necessary for running any web based application like WEBSTUIRE. Microsoft IIs is a free web server component. Also IIS consist of a set of Internet-based services for servers created by Microsoft for use with Microsoft Windows which turns the computer into a web server.

6.6.1.3 Development Language

Source codes of WEBSTUIRE are written with C# language. C# is a powerful and flexible language that it is suitable for developing web-based applications. Therefore using C# language, all of source codes required for developing WEBSTUIRE are written.
6.6.2 Implementation Environment

For implementation of WEBSTUIRE, some software and hardware requirements are needed that are explained from Table 6.1 Until 6.2:

Table 6.1 Software requirements needed for running WEBSTUIRE

<table>
<thead>
<tr>
<th>Type</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Operating System</td>
<td>Operating system that allow a browser to execute WEBSTUIRE application.</td>
</tr>
<tr>
<td>Microsoft SQL Database</td>
<td>Microsoft SQL Server 2000 to store all the WEBSTUIRE data.</td>
</tr>
<tr>
<td>Browser</td>
<td>Any browser such as Internet Explorer 5.0 and above and Firefox to access into web application system and can browse web base system.</td>
</tr>
<tr>
<td>Web Server</td>
<td>Microsoft IIs needs to install in order to execute WEBSTUIRE.</td>
</tr>
</tbody>
</table>

Table 6.2 Hardware requirements needed for running WEBSTUIRE

<table>
<thead>
<tr>
<th>Type</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer with wide band internet connection</td>
<td>Use to run WEBSTUIRE .</td>
</tr>
<tr>
<td>Computer keyboard</td>
<td>Allow user to input data into the system.</td>
</tr>
<tr>
<td>Mouse</td>
<td>Use to scroll the pages.</td>
</tr>
</tbody>
</table>
CHAPTER 7- COMPARISON AND EVALUATION

7.1 Comparison with Related Work

CASE tools in software engineering use different approaches and exhibit variety of capabilities. Some tools focus on the graphical aspects of design while others tend toward text aspect of requirement specification and UML modeling. No existing CASE tools and add-ons exactly resemble WEBSTUIRE, but there are CASE tools that seem to be similar in some aspects. On the other hand, these CASE tools differ in interesting ways with regard to capability or focal area. This chapter gives a comparison between WEBSTUIRE and other related tools. The goal of this comparison is to discover both strengths and weaknesses of WEBSTUIRE compared with other similar tools.

The CASE tools chosen for our comparison are listed below:

- DOORS with the Scenario Plus add-on
- OSRM (Requirements Management Tool)
- RUT (Requirements Use Case Tool)
- Enterprise Architect
- Altova UModel 2010
- Visual Paradigm for UML
- SUCRE
- STORM
- WEBSTORM
- SUIP tool
- SDE toolset

DOORS with Scenario Plus add-on are as powerful requirements tracking, managing, and traceability tools that are capable of extending in functionality and support misuse cases and export generated documents to Microsoft Word and PDF. The
first difference is that WEBSTUIRE generates scenarios from use cases automatically and provide capability to directly leave comments about scenarios for user in web interface. Furthermore, WEBSTUIRE uses the use case and scenario description based on templates proposed by Arlow and Neustadt (Arlow, 2002), whereas DOORS does not.

The Open Source Requirements Management tool (Smith, 2010) is a powerful requirement tool that is also open source. It has a lot of similar features as compared to DOORS. The capability of requirements priority and traceability matrix generation are the main difference with DOORS. The open source feature of OSRM makes it a good and cheap solution to manage requirements. It does not support use cases, scenarios and need to improve web interface. Lack of these features makes it different from WEBSTUIRE.

The RUT is a requirements use case tool with a web front-end that provides capabilities to create projects, actors, and use cases (RUT, 2010). It performs analysis of requirements and various computations on requirements such as path analysis and completion checks. An analysis of the text in use cases looks for keywords which determine a use case is too vague or incomplete. It is a powerful tool to use case alone. But, the scope of WEBSTUIRE differs from RUT in providing support for managing requirements, use cases, traceability matrix, sequence diagram, and the SRS document.

Another important category of CASE tools is diagramming editors such as Visual Paradigm (Visual Paradigm, 2010), Sparx System’s Enterprise Architect (Enterprise Architect, 2010), and Altova’s UModel (Altova UModel, 2010). They are popular CASE tools on the market that assist developers in software development. These tools are mainly concerned with the graphical aspect of the system and also are powerful diagramming tools which truly support visual design aspects. A considerable difference between WEBSTUIRE and them is that they do not handle the text aspects of
software specification, so may lose some specifications when only works with diagrams.

Visual Paradigm for UML (Visual Paradigm, 2010) is a popular diagramming tool that handles all UML diagrams. It offers extensive use case modeling features including full function UML use case diagram, flow of events editor, interaction and activity diagram generation. It also produces system documentation in XML and MS Word formats. But it does not support textual specification constructs or a web interface.

The significant feature of Enterprise Architect (Enterprise Architect, 2010) is excellent graphical capabilities and ability to do requirements modeling directly. The code generation capability is provided by integration with IDEs such as Microsoft’s Visual studio. Altova’s UModel is a designing tool that offers similar code generation capabilities in C#, Java, or Visual Basic.NET with reverse engineering support to generate UML diagrams from supported code base and vice versa. These tools have great features that distinguish them from other tools, but they do not handle requirement-based use case specifications in the way WEBSTUIRE does.

SUİP (SUİP, 2006) is a user interface prototyping tool developed by Elkoutbi et al. This tool applied a new approach to the generation of user interface prototypes from scenarios. The most important feature of this tool is that it derives the system user interface automatically. But it does not support the textual verification and specification of scenarios. WEBSTUIRE generates sequence diagrams enriched with UI information similar to SUİP does, but SUİP does not carry out the prototyping of user interface requirements in the way WEBSTUIRE does.

Scenario-based Model Driven Engineering Framework (Virani, 2008) is a framework that facilitates eliciting requirements. The important capability of SDE is that it receives XML format of sequence diagram as input and displays it in tree and
WEBSTUIRE also displays sequence diagrams enriched with UI information in tabular format. But SDE does not support another UML diagrams.

SUCRE is a Scenario and Use Case-based for Requirements Engineering framework (Alsomait, 2004). The framework enriched the Use Case Maps (UCM) with new visual notation for representing interface at different abstract levels. SUCRE is used to represent and validate user interface requirements while improving and mediating the communication between different parties involved. SUCRE builds operators to validate the UCM-UI model using design heuristics for constructing a formal specification. Moreover, part of SUCRE framework is a metrics suite to predict usability from scenarios and use cases. Likewise SUCRE bridges the gap between requirements and design. These tools have great features that distinguish them from other tools, but they do not handle scenario-based and use case-based for user interface requirements specifications in the way WEBSTUIRE does.

STORM is a stand-alone tool developed by Eric Fritzinger (2006) at UNR. The strength feature of STORM is its standalone approach without need for any plug-in or add-ons in the requirements modeling process. It focuses on text aspects of requirements specification. WEBSTORM (Talekar, 2008) is a new version of STORM that was re-designed and developed to provide all management capabilities of STORM in a web-based environment. However, WEBSTORM inherits many aspects from STORM. But it improved the use case and scenario generation, and SRS document generation, and also provides a simple graphical interface for creation of the traceability matrix. WEBSTORM’s architecture provides for a multiple user environment.

In summary, WEBSTUIRE was designed for text aspects of user interface requirements elicitation and specification with web based environment, which sets it apart from other CASE tools. The aforementioned tools have powerful features and different approaches in their respective areas, but do not carry out everything that
WEBSTUIRE does. Only a few tools come close to the approach proposed in WEBSTUIRE. Table 7.1 contains a chart comparing WEBSTUIRE with the CASE tools mentioned in this section under a number of criteria selected for comparison. Although these selected tools for reference have other strong features and capabilities that WEBSTUIRE does not focus on, the selected set of criteria used only serve to better characterize WEBSTUIRE among these CASE tools discussed.

Table 7.1 WEBSTUIRE Comparison with Similar CASE tools

<table>
<thead>
<tr>
<th>CASE Tool</th>
<th>Requirements modeling</th>
<th>Use case test input</th>
<th>Automated scenario generation</th>
<th>Full diagram capabilities</th>
<th>Web/Multi User Interface</th>
<th>Sequence diagram in tabular format</th>
<th>Automated UI screens generation</th>
<th>Automated Navigation Map generation</th>
<th>Mock-up and prototype generation</th>
<th>End-User involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEBSTUIRE</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>WebStorm</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>STORM</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DOORS with Scenario Plus</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>RUT</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Visual Paradigm</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Enterprise Architecture</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Altova UModel</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>OSRM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SUIP</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SDE framework</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>SCURE framework</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

7.2 WEBSTUIRE Evaluation and Results

In this section, the evaluation of developed system is done from view points of the users. For evaluating developed system, there are different types of users, developers and end-users, to use the system and examine some usability and functionality features of it. And then the evaluation results of WEBSTUIRE by users are displayed in diagrams so that the researcher is able to find out viewpoints of the users by analyzing users' feedbacks of the system.
7.2.1 Methodology of Evaluation

In order to evaluate WEBSTUIRE, a questionnaire form which considers some important critical functionality and usability features associated with the WEBSTUIRE was designed. Then questionnaire forms are filled up by users who used the WEBSTUIRE to evaluate it. Finally, the suggestions and recommendations of users in questionnaire were reflected in diagrams which they were based on. These questionnaires distributed to the ten users from different majors of postgraduate students in the faculty of Computer Science and Information Technology in UM who work as software developers at several companies.

7.2.2 Design of Questionnaire Form

Questionnaire form that is designed to evaluate WEBSTUIRE includes two parts. Part A of questionnaire concerns about personal information of participants, and part B contains two types of questions, the first type for evaluating system functionality and the second type for evaluating system usability.

The questions considered main important features concerned with functionality and usability of WEBSTUIRE. These questions are rated between one to five which consider one as poor, two as fair, three as good, four as very good and five as excellent. Appendix A shows the questionnaire form. The questions that defined to evaluate usability and functionality of WEBSTUIRE are listed as follow:

Usability of WEBSTUIRE measured based on:

1. WEBSTUIRE is easy to use.
2. WEBSTUIRE is easy to learn.
3. WEBSTUIRE handles error messages judiciously.
4. User interface of WEBSTUIRE is interactive and satisfaction.
5. Time of response of WEBSTUIRE is satisfying.
Functionality of WEBSTORM measured based on:

1. Creating project outline
2. Managing requirements in projects
3. Managing use case in project
4. Managing scenarios in project
5. Managing sequence in project
6. Analysing mockup in project
7. Generating UI-SRS

7.2.3 Evaluation of Results

When all of questionnaire forms filled by the participants are gathered, the analyzing of gathered result from viewpoints of the participants is started. Evaluation results are displayed the feedback of questionnaire forms in terms of charts as visualized in below.

Role of participants:

Figure 7.1 displays the roles of participants who participate in this tool evaluation process. As shown in Figure 7.1, 70% of the participants are as a software developer.
WEBSTUIRE Usability:

Figure 7.2 represents the results of usability evaluation of WEBSTUIRE by participants. Participants evaluated usability of the WEBSTUIRE in terms of five important usability features in questionnaire based on scores which explained one as poor, two as fair, three as good, four as very good and finally five as excellent. Figure 7.2 displays the average usability results of ten respondents.
WEBSTURE Usability

Figure 7.2 Average results of evaluating of WEBSTUIRE usability

WEBSTUIRE Functionality

Figure 7.3 represents the results of functionality evaluation of WEBSTUIRE. Participants evaluated functionality of the WEBSTUIRE in terms of seven important functionality features in questionnaire based on five scores similar to usability scores. Figure 7.3 displays the average functionality results of ten respondents.
Figure 7.3 Average results of evaluating of WEBSTUIRE functionality

- Generating UI-SRS: 3.3
- Analysing mockup: 3
- Managing sequence: 3
- Managing scenarios: 3.9
- Managing use case: 3.8
- Managing requirements: 4.1
- Creating project outline: 4.3
Chapter 8-FUTURE WORK AND CONCLUSIONS

This chapter concludes this thesis. It reviews the previous chapters that described different aspects of combination of Scenario-Based and Prototyping techniques for User Interface Requirement Engineering. The first chapter shows the objectives and goals of combination scenario-based and prototyping methods to develop the proposed tool and it continues with a discussion of some ideas for future work on developing the tool.

8.1. Objectives and Goals

This research stated the developing a Web based Support Tool for User Interface Requirements Elicitation (WEBSTUIRE) aimed to facilitate the process of requirements elicitation including User Interface requirements based on combination of scenario-based with prototyping techniques. The most important objective of developing WEBSTUIRE is to help software developers to acquire and analyze user interface requirements and enable to get feedback from end-users in order to bridge the gap between developer and end-user.

Hence in continuing this research, the researcher tried to develop a software support tool to facilitate application of a combination of scenario-based and prototyping techniques in eliciting and capturing user interface requirements. It investigated the generation of SRS via WEBSTUIRE aimed to elicit requirements of system including user interface requirements in a conceivable format. Finally, the researcher has achieved the objectives and goals of this research.

The average results of evaluating of WEBSTUIRE functionality were more than 3 up to 5. These results indicated use of the scenario-based and prototyping techniques which helped the process of user interface requirements elicitation, facilitated elicitation and validation of user interface requirements process with the usage of combining these
techniques. Likewise the good scores in user interface-SRS generation stated the communication between developers and users has improved by these combining these techniques. Finally based on the evaluation data of WEBSTUIRE, the researcher has answered the questions of this research.

8.2. Future Research Works

The researcher encountered some problems during this thesis that help to suggest directions in future works on WEBSTUIRE. As explained in previous chapters, WEBSTUIRE supports user interface requirements elicitation and SRS generation which includes user interface requirements; however, the researcher recognized that the main difficulty in the creation mockups process is on providing sequence diagrams which enriched UI elements in text based format. There is an essential need for applying several suggestions in order to improve generation of mockups and navigation maps. WEBSTUIRE is capable of being used as a user interface requirements elicitation tool. Furthermore in the future, WEBSTUIRE can be developed to integrate with other tools to generate more accurate and more complete prototyping of system to facilitate acquiring user interface requirements of system and generation of a complete SRS.

WEBSTUIRE can be improved by expanding the following features in the future works:

I. More diagram generation such as class diagram, activity diagrams, and so forth;

II. Improve the generation sequence diagrams that support other features of sequence diagram in UML2.

III. Navigation maps can form the detail of movement among the widgets of UI screen.
IV. Exporting WEBSTUIRE projects to XML formats to support integration with other CASE tools.

V. Improve generation of mock-ups and SRS document.

8.3. Conclusions

In this thesis, the main aim to put end-users at ease is bridge the gap between developer and end-user in user interface requirements engineering, and obviate the drawbacks of existing tools encourage us to develop WEBSTUIRE. So we reviewed several tools for using scenarios and prototyping in user interface requirement engineering. We discovered that few tools support the use of scenarios. Among these tools, WebStorm is deeply analyzed because WEBSTUIRE inherits many of its core features from WebStorm which is a CASE tool with web interface focused on the requirements elicitation and specification phases of the software engineering life cycle. WEBSTUIRE is based on the same scenario-based requirements process. First, it gathers the main requirements of the system, then extracts scenarios from use case models, and analyzes the scenarios and validates the requirements and use cases. This approach is a combination of Arlow and Neustadt’s guidelines (Arlow, 2002) which were modified by Ian Sommerville’s software engineering approaches (Sommerville, 2004). Additionally, WEBSTUIRE added sequence diagrams enriched with User interface elements to provide scenario screens and navigation map. The methodology is a combination of the Virani (Virani, 2008) and Elboukit (Elboukit, 2006) approaches to generating user interface prototype. The most important feature of WEBSTUIRE is that it takes into account end-users of the system during the requirement elicitation phase and can directly get feedback about each part of system being in progress, to provide more accurate user interface requirements.
In summary, WEBSTUIRE helps to improve the quality of user interface of system and training time because the system’s end-user has been given the opportunity to evaluate the system functionality and user interface during the earlier phase of software development. Finally it reduces the requirement changes in next phases of software development lifecycle which are expensive and time consuming.
References


121


APPENDIX A

Evaluation Questionnaire Form for
Web-based Support Tool for User Interface Requirements
Elicitation (WEBSTUIRE)

Part A: Personally Information

1- What is your age?  
   20-25 [ ]  26-30 [ ]  over 30 [ ]

2- What is your major?  
   MSE [ ]  MIS [ ]  MCS [ ]  MLIS [ ]

Part B: WEBSTUIRE Evaluation

{Poor [1], Fair [2], Good [3], Very Good [4], Excellent [5]}

Functionality of Tool

1. Creating project outline  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
2. Managing requirements in projects  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
3. Managing use case in project  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
4. Managing scenarios in project  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
5. Managing sequence in project  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
6. Analysing mockup in project  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
7. Generating UI-SRS  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]

Usability of Tool

1. WEBSTUIRE is easy to use.  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
2. WEBSTUIRE is easy to learn.  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
3. WEBSTUIRE handles error messages judiciously.  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
4. User interface of WEBSTUIRE is interactive and satisfaction.  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]
5. Time of response of WEBSTUIRE is satisfying.  
   1[ ]  2 [ ]  3 [ ]  4[ ]  5[ ]