

An Agilized Roadmap for User-Centered Requirements and Prototype Derivation

Muhammad Faraz Anwar

A Thesis

In

The Department

Of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master Of Computer Science at

Concordia University

Montreal, Quebec, Canada

March 2005

©Muhammad Faraz Anwar, 2005



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

ISBN: 0-494-04440-3

Our file *Notre référence*

ISBN: 0-494-04440-3

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

ABSTRACT

An Agilized Roadmap for User-Centered Requirements and Prototype Derivation

Muhammad Faraz Anwar

In interactive software system engineering, the user-interface requirement is a crucial phase. Practitioners are always looking for better guidelines and roadmaps for requirements engineering that not only ensure the usability in future system, but are also easy to master and use. This is one of the major challenges of the current HCI methods for user requirements. Agile methodologies are known for their ease of use, flexibility, and fast, iterative approach. The ideas from agile methodologies along with user-centered requirements engineering practices give a great motivation for a unified approach for user-interface requirements engineering. In this thesis, we propose a roadmap for user-centered requirements engineering that has agile characteristics. This roadmap, we believe, will be helpful in acquiring usability requirements and will be easy to follow. The major artifacts in this roadmap are Scenarios. A Prototype Derivation Process presented in this thesis complements the SUCRE framework, which was developed in our research group for scenario based requirements engineering. This process serves important milestones in the roadmap for quick requirements verification. The prototype thus created is in the form of storyboard and could be used for evaluation by end-users or developers. Therefore, the major contributions of this thesis are an Agilized Roadmap for User-Centered Requirements and process for Prototype Derivation from scenarios that are created as the major artifacts of the roadmap.

ACKNOWLEDGEMENTS

I am thankful to Almighty God for making me able to work on and complete this thesis. After that, I would like to express my sincere acknowledgements to several people whose role, direct or indirect, during the research for this dissertation was valuable.

My thesis supervisor, Dr. Ahmed Seffah, has been a guide, a coach, and a friend throughout this time. I am grateful for the great learning opportunities he provided me in terms of discussions and teaching assistantships. Many thanks to Dr. T. Radhakrishnan, Dr. O. Ormandjieva of thesis examining committee for feedback on this thesis.

I am also thankful to all the members of our Human-Centered Software Engineering group at Concordia University. Their assistance was available whenever needed. We made a warm and friendly team and a great learning environment.

And now thanks to those who made great indirect contributions to this research: *Ejaz bhai*, *Surayya baj* and family; they have been great comforters for me during all my time in Canada. Thanks to my new friends in Montreal, and above all, many thanks to my parents and family in Pakistan and love to *Hamna* and *Bilal*, being eager to meet them after finishing this thesis kept me working incessantly.

TABLE OF CONTENTS

LIST OF FIGURES	x
LIST OF TABLES	xii
Chapter 1	
Introduction	1
1.1 Research Objectives.....	5
1.2 Research Methodology	6
1.3 Thesis Organization	8
1.4 Basic Definitions and Terminologies.....	10
Chapter 2	
An Analysis of Agile and User-Centered Requirements Engineering...	13
2.1 Introduction.....	13
2.2 Agile Software Development.....	15
2.2.1 Highlights of Agile Development Philosophy	16
2.2.2 Focus on Major Agile Methods	19
2.3 User-Centered Design Philosophy	24
2.3.1 Highlights of the User-Centered Design Philosophy	24
2.3.2 Focus on Major UCD Methods.....	26
2.4 Discussion: Commonalities and Differences	30
2.5 Brief Introduction to the Treatment of Requirements in UCD and Agile Methods	33
2.5.1 Requirements Engineering in User-Centered Design.....	33

2.5.2 Requirements Engineering in Agile Methodologies.....	36
2.6 An Overview of SUCRE Framework	38
2.7 Conclusions.....	41

Chapter 3

An Agilized Roadmap for User-Centered Requirements Engineering . 42

3.1 Introduction to Roadmap	42
3.1.1 Motivations for the Roadmap	42
3.1.2 Benefits of Roadmap Approach.....	43
3.1.3 Roadmap versus process and methodology	43
3.2 The Proposed Roadmap	44
3.2.1 Notation used in the Roadmap.....	44
3.2.2 Structure of the Roadmap	45
3.2.3 Description of milestones in the Roadmap	47
3.3 Development of the roadmap.....	54
3.4 Discussions about the roadmap.....	59
3.4.1 When to Use the roadmap?.....	59
3.4.2 Agile Aspects in the Roadmap.....	59
3.4.3 Scenarios as the Major Artifacts of the Roadmap	61
3.4.4 Basis and Validity of the Roadmap	65
3.5 Conclusions.....	67

Chapter 4

Scenarios and their Representation as Use-Case Maps 68

4.1 Introduction.....	68
-----------------------	----

4.2 Use-Case Maps as Scenario Representation	70
4.2.1 The Basic Notation	71
4.2.2 Advantages of UCMs.....	73
4.3 Use-Case Maps in SUCRE framework.....	75
4.3.1 Advantages of SUCRE framework.....	79
4.4 Conclusions.....	81

Chapter 5

Prototype Derivation from Scenarios	82
5.1 Motivations for a Prototype Derivation Process.....	82
5.2 Prototyping in the Agilized Roadmap.....	84
5.3 Prototyping in SUCRE Framework	85
5.4 Process of Prototype Derivation from PUCMs.....	87
5.5 Extension to Presentation Units (PUs).....	89
5.5.1 Rationale for extension of PU Symbols.....	89
5.5.2 Some Presentation Units for PUCMs	91
5.6 Prototype Elements	92
5.6.1 Development for PE Symbols.....	92
5.6.2 Some PE Symbols for Prototyping	94
5.7 Mapping Matrix - A tool for Deriving Prototypes using UCMs	97
5.7.1 The proposed PU-to-PE Mapping Matrix.....	97
5.8 Concluding Remarks about the Mapping Matrix.....	101

Chapter 6

Illustration of the Prototype Derivation Process	102
---	------------

6.1 Introduction.....	102
6.2 An introduction to dashboards and GPS.....	103
6.2.1 Scenarios related to dashboards and GPS.....	104
6.3 Step 1 – Scenario Elicitation.....	105
6.3.1 Background of the Scenario.....	105
6.3.2 The Scenario	105
6.3.3 Capturing Tasks/Actions from Scenario.....	106
6.4 Step 2 – Scenario Specification	108
6.4.1 Use-Case Maps	108
6.5 Step 3 – Scenario Analysis	112
6.5.1 Applying Mapping Matrix	113
6.5.2 Storyboard Prototype	115
6.5.3 Explanation of the Storyboard	118
6.6 The storyboard and UI requirements in scenario.....	119
6.7 Conclusion of Illustration	120
Chapter 7	
Conclusions and Suggested Future Research	121
7.1 Major Contributions.....	123
7.2 Suggested Future Research.....	125
References.....	127
Appendix A	
Forms and Templates for the Roadmap.....	134
A.1 Vision Document	135

A.2 Forms for Field Study	136
A.2.1 Identifying Users and Stakeholders	136
A.2.2 User Characteristics	139
A.2.3 Social environment of use.....	140
A.2.4 Users' Anecdotes	142
A.3 Usability Goals.....	144
A.4 Stating Scenarios.....	146
A.5 Task Analysis.....	147
A.6 Study of Similar Systems.....	150
A.7 Physical Environment	151

Appendix B

List of Suggested Presentation Units (PUs).....	153
B.1 A Note about Presentation Units.....	154
B.2 Presentation Units	154

Appendix C

List of Suggested Prototype Elements (PEs)	159
C.1 A Note about Prototype Elements.....	160
C.2 Prototype Elements	160

LIST OF FIGURES

Figure 2.1: Evolution of SUCRE framework	38
Figure 3.1: Structure of the roadmap	46
Figure 3.2: First version of the roadmap.....	55
Figure 3.3: Second version of the roadmap	56
Figure 3.4: Third version of the roadmap	58
Figure 3.5: Typical steps in scenario creation	63
Figure 3.6: Integration of UCD practices in RUP (Anderson et al.).....	66
Figure 4.1: Simple UCM for ATM login.....	71
Figure 4.2: Decomposition of ATM Console into two components.....	72
Figure 4.3: An Unbound UCM	72
Figure 4.4: Simple UCMs for ATM and Web-Application login.....	74
Figure 4.5: SUCRE framework (Alsumait, 2004)	75
Figure 4.6: Dialog symbols for CUCMs (Alsumait, 2004)	76
Figure 4.7: Presentation Units (PUs) suggested by (Alsumait et al., 2003)	78
Figure 5.1: Prototyping in SUCRE framework (Alsumait, 2004).	85
Figure 5.2: Process of Prototype Derivation from Scenarios through UCMs	88
Figure 5.3: Initial car-dashboard widget hierarchy	93
Figure 6.1: GPS in a car dashboard.....	103
Figure 6.2: Conceptual Use-Case Map for Scenario	109
Figure 6.3: Physical Use-Case Map for Scenario	110
Figure 6.4: Application of Mapping Matrix	112

Figure 6.5: Suggested Prototype Elements (PEs) for Presentation Units (PUs)..... 114

Figure 6.6: Screens of Prototype..... 116

Figure 6.7 Final Storyboard 117

LIST OF TABLES

Table 2.1: Extreme Programming – Major concepts and practices	20
Table 2.2: Crystal Methods – Major concepts and practices	21
Table 2.3: DSDM – Major concepts and practices	22
Table 2.4: Scrum – Major concepts and practices	23
Table 2.5: Scenario-based design – Major concepts and practices.....	27
Table 2.6: Contextual Design – Major concepts and practices.....	28
Table 2.7: Usage-Centered Design – Major concepts and practices	29
Table 2.8 Summary of Commonalities and Differences in agile and UCD philosophies	32
Table 3.1: Symbols used in roadmap.....	45
Table 3.2: Agile aspects in roadmap.....	60
Table 5.1: Examples of Presentation Units.....	91
Table 5.2: Examples of Prototype Elements.....	96
Table 5.3: The Mapping Matrix.....	100
Table 6.1: Analysis of Scenario	107
Table 6.2: Applying Mapping Matrix to PUCM	113

Chapter 1

Introduction

User-centered requirements engineering is a significant field in software engineering. It is primarily users' requirements that the software aims to fulfill. Requirements engineering is the topic of study for many researchers (Carroll et al., 1998, Maguire, 1998, Orr, 2004, Paetsch et al., 2003). In this thesis, we discuss the area of requirements engineering as practiced in User-centered software engineering community. We develop methods to efficiently elicit and specify users' requirements and to represent them in terms of scenarios and using scenarios to develop prototypes.

One common aspect in almost all interactive software systems is that they should support human experiences while providing task- and context-aware interaction. Designing such interactive systems is no trivial task; therefore, trial-and-error is not sufficient and there has to be a well-defined method for requirements engineering. Practitioners and researchers of requirements engineering work together to devise methods to ensure that product has good quality and satisfies the purpose it was meant for. The method must also be usable by software engineers.

Requirements are collected from the domain the software is to be used in. Besides all the current methods that exist, one thing that should be of principal concern in requirements engineering methods is the focus on end-users. Orr argues that "the requirements

engineer's job is to help the users discover what they really need" (Orr, 2004). Since the product is made for human users, its success depends upon how well humans can use the system. The focus of requirements engineering and design in User-Centered Design and User-Centered Requirements Engineering is user and not the functionality (Lauesen, 1997).

There are two levels of human interaction involved in user-centered requirements engineering methods. First, the requirements engineers use these methods to interact with end-users to collect domain knowledge, and secondly, since the methods are used by human software engineers, the process itself must also be easy for humans to be followed. This is where agile software methodologies come into play. Agile development is a well-established idea practiced by many people in software engineering community (e.g. (Kutschera and Schafer, 2002, Fowler, 2000, Beck et al., 1998)). Its methodologies, for example Extreme Programming, advocate the involvement of users during the software development and flexibility in the process to adapt with human work psychology. Agile aspects in user-centered requirements engineering process, we believe, can give a solid and comfortable ground for user-centered requirements engineers.

Current agile methodologies, however, focus only on the development phase. Team collaboration, iterations, short deliveries etc. are all practical in implementation phase in the agile domain. Therefore, we have to adopt the agile philosophy and its key concepts in the initial requirements engineering phase so as so facilitate the requirements gathering process and to enrich it with ideas from agile philosophy. We have proposed in this thesis

a roadmap for user-interface requirements engineering where we have taken ideas from agile methodologies. This roadmap serves as a link between UCD and agile worlds.

Scenario elicitation is a key step in the *agilized* usability requirements engineering roadmap presented in this thesis. Scenario elicitation and analysis is itself a complex task and need to be explained so that the roadmap could be followed well. For this, we advocate the use of SUCRE framework (Alsumait, 2004). SUCRE is an extensive framework for defining and representing scenarios on conceptual and physical level.

Using this framework, we have devised a method to transform scenarios into prototypes. This is a tool in the form of a matrix that suggests prototype elements (figures that constitute a paper prototype) on the basis of symbols used in Physical Use-Case Maps (a term introduced in SUCRE framework that we use as a precondition for deriving prototypes). The prototype created using UI symbols suggested by the mapping matrix is closer to real UI and summarizes UI elements in terms of tasks they perform.

Prototype derivation is an important piece in requirements engineering puzzle and provides a great support in the roadmap we present. The prototype thus created could be used in storyboarding and helps verify the design at early stages in development lifecycle. Requirements can be verified by users through prototypes which are easy to understand by everyone involved in the process due to the standardized symbols proposed by our proposed mapping matrix. The SUCRE framework mentioned the prototype derivation from UCMs but did not give a clear-cut method to do so. The framework is thus

complemented with a tool based on the concepts proposed in the framework and therefore completes a Scenario-to-prototype path.

The two major contributions of this thesis are the *agilized requirements engineering roadmap* and a *tool for prototype derivation from scenarios*. These two ideas are linked with the SUCRE framework. They make up an important contribution in scenario-based usability requirement engineering.

1.1 Research Objectives

There are two main objectives of this research. The first objective is to find a common ground in User-Centered Design and agile philosophy. We strongly believe that UCD is an important facet of software engineering and must not be ignored even in the initial stages of software lifecycle. Requirements engineering is a vital pre-phase of UCD which helps makes sure that before starting user-centered design, the requirements must be collected in a way which is also user-centered. We observed that agile philosophy is a growing interest in software engineering circles and gives good ideas that we can incorporate into the user-centered requirements engineering. Therefore, we propose a unified roadmap with the best of UCD and agile methodologies.

The second objective is to support the major milestone of roadmap, namely Scenario Elicitation, such that prototype derivation is possible from scenarios. This objective also complements the SUCRE framework with a prototype derivation process. SUCRE (Scenario-based User-Centered Requirements Engineering) is a requirements engineering framework that has its foundations in scenarios and Use-Case Maps. In this framework, prototype derivation is a logical sequence after making Conceptual and Physical Use-Case Maps. With our second objective achieved, an important artifact in the SUCRE framework will be made possible.

1.2 Research Methodology

We started our research with doing a literature study of UCD and agile philosophies. We explored several methods proposed under these two notions and tried to find common grounds in them. We also covered papers over requirements engineering and its role in UCD and agile methodologies.

We then made a roadmap which lays out milestones for gathering user-interface requirements and has concepts of agile philosophy working behind the scene. We name this road map as *An Agilized Roadmap for User-Centered Requirements*.

We focused on Scenario Elicitation milestone in this roadmap and studied papers and other material about scenarios. We investigated the role of scenarios in requirements engineering, different modes of representation of scenarios and found Use-Case Maps (Buhr, 1998) as most fitting for our need. We studied SUCRE framework (Alsumait, 2004) which is developed in our research group at Department of Computer Science, Concordia University. The SUCRE framework recommends expounding scenarios in two tiers of abstraction: Conceptual and Physical. Physical UCMS follow the Conceptual UCMS in order of creation and have decreasing levels of abstraction. Physical UCMS are closer to user-interface setting but still is too abstract to be worked on. The logical consequence of physical UCMS is prototypes which are much closer to real user-

interfaces. We thus formulate a tool for suggesting elements of prototype on the basis of elements of physical UCMs.

This two-step approach (roadmap and tool) enables us to see a big picture of requirements engineering process and gives us practical ideas about conducting requirement engineering.

1.3 Thesis Organization

This thesis is organized into seven chapters including this one. The organization of the chapters is as follows:

Chapter 1: *Introduction* gives introduction of the thesis and how we came up with the ideas presented in this thesis. It also outlines the research objectives and methodology.

Chapter 2: *An Analysis of Agile and User-Centered Requirements Engineering*. This chapter includes the result of our literature study on the subjects of User-Centered Design and agile methodologies. We first introduce highlights of the philosophies and then present some methods that are used in industry and hold the flag of their respective philosophies. A discussion about the common and dissimilar points in these notions followed the introduction of methods.

Chapter 3: *An Agilized Roadmap for User-Centered Requirements Engineering*. In this chapter we introduce the roadmap we developed iteratively based on the ideas of UCD and agile philosophies. After the roadmap, there is description of milestones of the roadmap and discussion about its benefit and agile characteristics.

Chapter 4: *Scenarios and their Representation as Use-Case Maps*. Here we have detailed discussion about scenarios and their use in requirements engineering. We also present

Use-Case maps which are a way to represent scenarios. Later on, we introduce the user-interface extension to UCMs and how this enrichment could be used to represent scenarios related to user-interfaces.

Chapter 5: Prototype Derivation from Scenarios. The process of deriving prototypes from scenarios is presented in this chapter. We first discuss about prototyping in the roadmap (presented in chapter 3) and in SUCRE framework. Then we explain how the prototype derivation works based on scenarios. Later on, we introduce necessary extensions to the notation for Physical UCMs and the set of symbols that is to be used in prototypes. We then present the Mapping Matrix, a tool to suggest prototype elements from presentation units (of physical UCMs).

Chapter 6: Illustration of the Prototype Derivation Process. In this chapter, we illustrate the process of prototype derivation from scenario. We present a simple scenario and after analyzing it carefully, we apply the prototype derivation process (defined in chapter 5) to get a storyboard prototype.

Chapter 7: Conclusions and Suggested Future Research. In this last chapter, the whole thesis is summarized and major contributions of the research are highlighted. Suggestions for future research are later mentioned in the last subsection.

1.4 Basic Definitions and Terminologies

Following terms and phrases appear frequently in the text of this thesis with their respective meanings in mind. The meanings stated here will help in better understanding of the thesis.

Agilized – Having characteristics of agile software development. Typical characteristics of agile software development are less involvement of deliverables, fast and frequent deliveries, iterative, customer collaboration etc.

Presentation Unit – Icons representing parts of user-interface in physical use-case maps (PUCM). The responsibilities (actions) that user performs to accomplish task in a scenario are denoted by a cross over a Presentation Unit meaning that those actions will be performed on certain parts of user-interface represented by that PU.

Process – A process is a specification of sequence of steps that, when followed, result in achieving a particular goal or producing a product. In our thesis, for example, we propose a process for deriving prototypes from scenarios.

Prototypes – Artifacts representing a product under development with varying degree of abstraction (also called '*fidelity*') to show certain features of the product. Traditionally,

three types of prototypes are identified as: low-fidelity (e.g. using paper-pencil), mid-fidelity (e.g. storyboards) and high-fidelity (e.g. a working web-interface).

Prototype Elements – Graphical elements of a storyboard prototype screen. The Prototype Elements (or PEs) represent widgets of an interface in their simple forms i.e. not showing detail of the behavior of the widget. Users can understand the screen layout and what actions could be performed on the screen using the widgets that are represented by PEs.

Roadmap – A detailed plan to guide progress toward a goal (Merriam-Webster, 1982). It is an explicit formal specification of how to represent the objects, concepts and other process entities that are assumed to exist in user-centered requirements and the relationships that hold among them.

Scenarios – Stories in simple natural language (primarily English) which tells about a typical user of a system who is performing some tasks using a system under typical context of the environment.

Storyboard – A sequential series of sketches/pictures that show the flow of a story. In terms of scenario-based usability requirements engineering, the story is a scenario and the storyboard is a series of pictures depicting user-interface screens.

SUCRE – Acronym for Scenario-based User-Centered Requirements Engineering. It is a requirements engineering framework that is based on scenarios as major artifacts. The framework uses Use-Case Maps to represent scenarios. It was proposed by Alsumait (Alsumait, 2004) at Concordia University, Montreal, Canada.

Use-Case Maps – A notation to represent behavior of systems in a high-level way. Each ‘map’ corresponds to ‘use cases’ which relate to a particular function of a system. The notation was invented by Buhr (Buhr, 1998) at Carleton University, Ottawa, Canada.

User-centered requirements engineering – An approach involving potential users of a system under development for gathering requirements. This is in contrast to conventional requirement engineering approaches where the focus is on functions/features and users are involved at later stages or not involved at all.

Chapter 2

An Analysis of Agile and User-Centered Requirements Engineering

2.1 Introduction

There are numerous methodologies and processes that govern the software development. Every process has its own features and some overlapping characteristics with other processes. This overlapping is due to the fact that there are some philosophies behind them. Most of the methodologies we see today are based on a certain way of thinking which dictates the key concepts and practices of those methods.

In software development, there are two prominent philosophies that consider human/user involvement: *User-Centered Design* and *Agile Software development*. Both of these philosophies give a particular way of thinking about software engineering. Although these are two different philosophies, but we can draw some parallels between them and try to find a method that has best of both worlds.

In this chapter, we will shed light on the two philosophies and present some of the methods that these two philosophies govern. This practice will help us to understand the philosophies in detail and how the methods are affected by a way of thinking. We will later discuss the treatment of requirements in these two philosophies, which is our main goal in this chapter. A brief introduction to some requirements engineering methods will then be given.

2.2 Agile Software Development

The Software Development Process has undergone numerous revolutions since its inception. One of these revolutions is emergence of the philosophy of Agile Software Development. This philosophy is based on the notion that software development teams are focusing more on creating useless documentation and on the process itself rather than focusing on the product. The result is more delayed or failed projects. Agile philosophy and its supporting methodologies make sure that the development process is free from less fruitful rituals found in earlier processes. It gives new ideas for improving the communication between the team members and avoiding loopholes in development.

Agile philosophy took its current form with the emergence of agile manifesto in 2001 (Beck et al., 2001). A group of practitioners came together to discuss new processes that were not *heavyweight* or documentation-oriented. What they came up with was a set of following values:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

This philosophy and related methodologies have created lots of interest in professionals and academia. Abrahamson and others (Abrahamson et al., 2002) have discussed major agile methods with respect to *Process, Roles and Responsibilities, Practices, Adoption and Experiences, Scope of use*, and finally *Current Research*. As a result of this approach, they have presented a definition and classification of agile methods, and different methods are compared with each other with respect to these aspects.

The most important work that was needed to be done, and was attempted by many researchers, is the adoption of agile values in conventional software engineering practices. Kutschera and Schafer (Kutschera and Schafer, 2002) have presented a way to adopt agile methods in dynamic environments. Paetsch and others (Paetsch et al., 2003) have analyzed the role of agile methods in requirements engineering. In this thesis, we also have presented a roadmap for requirements engineering that confirms with agile values.

2.2.1 Highlights of Agile Development Philosophy

By definition, agile means: *marked by ready ability to move with quick easy grace and/or having a quick resourceful and adaptable character* (Merriam-Webster, 1982). The agile software development philosophy perfectly agrees with this definition. The philosophy advocates that the development process must always be ready to welcome change and yet must move with quick pace. The fruit of this thinking is more satisfied customers, developers with friendly rapport, and above all, good working software.

Most of the literature about this philosophy is produced by practitioners and consultants. As a result, this literature focuses on methodologies. Methodologies impose a disciplined process over software development with the aim of making development predictable and efficient (Fowler, 2000). However, the *Agile Manifesto* (Beck et al., 2001) gives a solid philosophical ground for methodologies. According to the manifesto, Agile Software Development is based on four values:

1) Individuals and interactions over processes and tools

Agile philosophy has a people-first orientation for software development (Abrahamson et al., 2002). That is, people are more important than processes. The software development process must suit the individuals who are developing the software. Some processes are better adapted by a group of developers in one culture but does not so in another culture or environment. According to Cockburn (Cockburn, 2000), people should not be treated as *components* that program. Rather, people are thinking and communicating beings suited for face-to-face communication. Therefore, one important breakthrough in agile methodologies is the importance of working with programmers' instincts through verbal communication (two-person teams in Extreme Programming, scrums in Scrum etc).

2) Working software over comprehensive documentation

Customers are always concerned with working software and have little interest in long documentations. Therefore, agile philosophy emphasizes on short but quick deliveries of working software. This does not mean that it discourages any kind of documentation, rather, the documentation should be done but only late in the process and when needed.

The lack of documentation is the indication of two built-in characteristics of agile methods: (1) Agile methods are adaptive rather than predictive; i.e. they welcome change and also can change themselves according to the situation. (2) Agile methods are people-oriented rather than process-oriented; role of process is to support people (teams) in work (Fowler, 2000).

3) Customer collaboration over contract negotiation

Although contracts are important from business point-of-view, they should not become a barrier against the communication between two parties. Agile philosophy ensures that the development team and client should collaborate with each other, especially over the requirements and do not freeze the requirements in the beginning of the project (this is particularly good for clients with changing requirements).

4) Responding to change over following a plan

Requirements change during the course of project. This fact has been taken graciously by agile philosophy and provided this important value in its manifesto. One way to control unpredictability due to changing requirements is 'iterations'. The length of iteration matters and dictates how often this change will be accommodated into design. XP and Scrum, including other methods, advise about the iteration length (Fowler, 2000).

2.2.2 Focus on Major Agile Methods

In this section, we will focus on some of the major agile methods that are used in industry and have been most commonly studied. This will exemplify the highlights of the agile philosophy put forth in previous subsection.

Following agile methods are discussed in this section:

- Extreme Programming
- Crystal Methods
- Dynamic system development method
- Scrum

1) Extreme Programming (XP)

Extreme Programming is the most popular agile methodology. It is based on four values namely: *communication, simplicity, feedback and courage*. Based on these values, about a dozen practices are suggested. These practices are not new; they are tested, tried but forgotten. XP offers a lifecycle process with phases: Exploration, Planning, Iterations to release, Production and finally death phase (when there is nothing more to implement). It is aimed for small and medium sized teams. Stress is put on team work and empowering developer to make decisions. (Wells, 2004, Abrahamson et al., 2002, Fowler, 2000).

Following table (Table 2.1) summarizes the key concepts in XP and names and descriptions of major practices (Abrahamson et al., 2002).

Key Concepts	
Respond to changing customer requirements	
Groupware-style development	
Communication, simplicity, feedback and courage	
Major Practices	Description
Planning game	Programmer decides effort, customer decides time for releases
Small releases	At least once every 2 to 3 months
Metaphor	A shared story guiding the development
Simple design	Design is simplest possible for implementation
Refactoring	Code is reviewed removed to discrepancies
Pair Programming	Programmers are always paired in a team of two.
Collective ownership	Anyone can change any part of code anytime
Continuous integration	A new piece of code is integrated into existing code as soon as it is ready.
40-hour week	Programmers work for no more than 40-hours per week.
On-site customer	A representative of customer is always present on programming site.
Coding standards	Coding rules and conventions exist and must be followed by all programmers.

Table 2.1: Extreme Programming – Major concepts and practices

2) Crystal Methods

Crystal is a family of number of methods, plus the principles of tailoring these methods according to need. Among the four color-coded (Clear, Yellow, Orange, Red) ranges of methods, an appropriate method is chosen based on size and criticality of the project. Clear methods allow multiple teams to work in parallel; but these teams should be located in shared office-space. Life-critical projects are not suitable to be developed with Crystal methods. (Abrahamson et al., 2002)

The table 2.2 highlights key concepts and major practices of Crystal methodologies.

Key Concepts	
Tailored process for every project	
Strong, face-to-face and short-path communication	
Less deliverables	
Major Practices	Description
Annotated usage scenarios	Scenarios are created and annotated.
Regression testing frameworks	Supports regression testing.
Peer code review	Programmers review code for each other
User involvement	User is involved in development.

Table 2.2: Crystal Methods – Major concepts and practices

3) Dynamic system development method

According to Abrahamson and others (Abrahamson et al., 2002) “The fundamental idea behind DSDM is that instead of fixing the amount of functionality in a product, and then

adjusting time and resources to reach that functionality, it is preferred to fix time and resources, and then adjust the amount of functionality accordingly.” A key concept of DSDM is “timebox” and each iteration must end in that timebox. Different phases of the DSDM are: Re-Project, Feasibility study, business study, Functional model iteration, Design and built iteration, Implementation, and lastly, Post-project phase. The team size for DSDM varies between two and six members. (Abrahamson et al., 2002)

Key concepts and major practices in DSDM are listed in Table 2.3 below.

Key Concepts	
Timeboxing (for timing of process)	
MoSCoW (for prioritizing tasks)	
Active user involvement	
Empowered team to make decisions	
All changes are reversible	
Major Practices	Description
Iterative and incremental development	Several iterations before final release
Testing throughout lifecycle	Every part of code is tested frequently

Table 2.3: DSDM – Major concepts and practices

4) Scrum

Scrum does not define any specific technique for implementation but gives a framework for flexibility, adaptability and productivity. It focuses on the changing values of variables that impact software quality and development process. These variables include

requirements, time-frames, resources and technology. A Scrum process includes three phases: Pre-game, Development, and Post-game. In the development phase, the system is implemented in *Sprints*, which are short iterations of development. The requirements are stored in a *backlog list*. A daily meeting is an important part of the development process, each meeting is known as *Scrum*. (Abrahamson et al., 2002)

Following table (Table 2.4) points out key concepts and major practices in Scrum.

Key Concepts	
Focus on team-work	
Daily communication of status	
Major Practices	Description
Sprints	Short deliveries
Scrums	Daily meetings

Table 2.4: Scrum – Major concepts and practices

2.3 User-Centered Design Philosophy

In any process of Software Engineering, *design* is an important phase. In this phase we consider the possible solutions of the problem, which was analyzed in *analysis* phase, and how to derive those solutions. During the design of the software, if we consider user as the focus of every activity, it can somewhat guarantee that the end product will be liked by users. User-Centered Design approach advocates the same idea that since users are the ultimate goals of software, their role should be incorporated into the design process right from the beginning to the end.

To support this idea of user-centered design and to give solid guidelines that can fulfill this purpose, different researchers have devised several methods (e.g. (Constantine and Lockwood, 2002)). These methods are based on a few *key-concepts* and advice some *practices* that will help in achieving a user-centered design. These key concepts also give interesting insights into how ideas from other Software Engineering practices and other fields like psychology could be adopted for a User-Centered Design.

2.3.1 Highlights of the User-Centered Design Philosophy

The philosophy of User-Centered Design and HCI has roots in disciplines of psychology, sociology, industrial design, graphic design and others. This amalgamation of paradigms has made UCD an interesting field. In software engineering, this is taken on purely

engineering approach and several methods are derived from this philosophy that makes the software development closer to user needs.

According to Xavier Ferre (Ferre, 2003), the iterative approach in UCD philosophy is crucial. It is impossible to make a correct design in first attempt due to the complexity of human behavior. Iterations, therefore, play a key role in defining user needs and refining them to render them useful.

The most obvious highlight of this philosophy is active user involvement. Unless the user is involved from the start of the software development process, it is difficult to make a system that is completely user-satisfying. The UCD philosophy enjoins the development team to contact user on each and every step of the process, get their feedback, inform them of the status of the progress, and above all, evaluate short deliveries with them.

One important concept in UCD philosophy is proper understanding of user and tasks (Ferre, 2003). Understating users is quite obvious, but for tasks, the UCD philosophy says that these are also important to understand. The viewpoint to look at tasks, in case of UCD, is different. In conventional methods, tasks are looked upon as *features to implement*. In UCD, tasks are set of actions that a user has to perform to achieve a goal. The viewpoint, thus, has shifted from system/software to user/human. This important shift in paradigm has enabled developers and designers to put themselves in user's shoes and see what user will have to do for hitting that goal. They thus design systems that are close to user's expectations from the system.

Users are humans. Humans are affected by their environment, so do users. The UCD philosophy also emphasizes the need to study user's environment and take decisions accordingly. Context is defined as the surroundings of users while they are using the system. Contextual inquiry is thus deemed important in UCD philosophy. Users' detailed sketch includes their education, exposure to similar systems, social status etc. make another important factor in their behavior with the system. UCD thus underlines the importance of understating the users themselves.

2.3.2 Focus on Major UCD Methods

User-Centered Design is a topic of research of a great many software engineering scholars and practitioners. They study topics from conventional software engineering, as well as different other fields, plus, come up with ideas of their own and formulate new methods that contribute in one way or the other to the vast area of User-Centered Design.

Here in this section, we will consider following three major UCD methods to give an idea of how their key-concepts and major practices can make the process and thus the product closer to the needs of end-users.

- Scenario-based design
- Contextual design
- Usage-centered design

The reason we have chosen these three methods is their relevance with our requirement engineering scope of study. Contextual design is one of the major methods for collecting requirements. Scenario-based design and Usage-centered design deals with Scenarios and Use-cases which are relevant to coming chapters in this thesis.

1) Scenario-Based Design

Scenarios are short stories telling about the use of system by the users. The Scenario-based design puts scenarios in focus and derives solution based on requirements gathered from them. In scenario-based design, descriptions of how people accomplish tasks are a primary working design representation (Carroll, 2000). Collecting scenarios involves users in telling stories about their use of the system. To collect and elicit scenarios, pictures, videos, and storyboards are used. Table 2.5 summarizes key concepts and major practice in scenario-based design.

Key Concepts	
Scenario	
Video, pictures, storyboards	
Major Practices	Description
User Involvement	Involve users to make and refine scenarios

Table 2.5: Scenario-based design – Major concepts and practices

2) Contextual Design

In this method of User-Centered Design, users are studied in their own environment while using the system. The designer goes to the field and observes how user interacts with the current system, what are different factors that affect user's behavior and affect system performance. Usability engineers take notes while observing and conduct user interviews. The impact of this method is reengineered task organization and task sequence models, in contrast to the implementation architecture in traditional systems analysis (Mayhew, 1999).

Highlights of Contextual Design are stated in Table 2.6 below:

Key Concepts	
Users and their work <i>in context</i>	
Data-gathering and Data-interpretation	
Major Practices	Description
Contextual Inquiry	A type of interview to gather field data from users.
Field Studies	Visiting places where system is used. Validate the information gathered from
Concept Validation	Contextual Inquiry

Table 2.6: Contextual Design – Major concepts and practices

3) Usage-Centered Design

In Usage-Centered design, the focus shifts to usage, rather than user. The process is driven by models based on usage of system. This systematic process uses abstract models to design small and simple system that fully supports all the tasks users need to accomplish (Constantine and Lockwood, 2002).

The following table lists key concepts and major practices of Usage-Centered design (Constantine and Lockwood, 2002).

Key Concepts	
Streamlined process driven by simple models:	
Role-models	
Task-models	
Content-model	
User-roles	
Major Practices	Description
Exploratory modeling	Identifying questions or areas of ambiguity.
Design-by-modeling	Making three models to drive the design: Role model, Task Model and Content Model
Card-based modeling	Simple inventory of roles users can play

Table 2.7: Usage-Centered Design – Major concepts and practices

2.4 Discussion: Commonalities and Differences

As we have seen in preceding section, UCD and agile software development are two different philosophies, developed by different people at different times. Yet, they have many aspects in common. In this section, we will shed some light on the commonalities and differences between these two philosophies. The summary of the following discussion is presented in Table 2.8.

Two values of agile manifesto are: (1) Individuals and Interactions over processes & tools. (2) Customer collaboration over contract negotiation. These values are in harmony with the UCD concepts of putting the emphasis on individuals (users and stakeholders). Stakeholders are people who have any interest in the software. The end-users are one of these stakeholders. In agile, any stakeholder (called *Customers*) is given same importance and is encouraged to interact with the development team.

On the other hand in UCD, the end-user is the primary concern of the usability team since it is the end-user who is going to interact with the user-interface of the software. In the context of agile methodologies, Individuals also refer to development team members with different skill-sets.

During discussion about the role of overlapping lifecycle phases, Mayhew (Mayhew, 1999) points out that optimal implementation of the lifecycle requires full participation of all teams. In traditional software engineering, however, people of different skill-sets work on their own part of lifecycle, and communication is done through documents.

Instead, if all people work together in each phase of the project, they can input their expert advice and raise their concern at the right time. This idea of collaboration in Usability lifecycle resonates perfectly with these agile values.

The other two values of agile philosophy are: (1) Working software over comprehensive documentation, and (2) Responding to change over following a plan. These values are not very common in UCD circles. In UCD, emphasis is put on getting the user-goals and requirements in written form. Style-guides are suggested to be made/updated after every major phase (Mayhew, 1999). Prototypes are encouraged to be made and evaluated long before the actual product is produced.

In agile methodologies, a working, deliverable version of software is always desirable and documentation is delayed to be done as late as possible. Change tolerance is also projected in UCD, but responding very quickly to change sacrificing the process is not advocated. Rather, this change management is incorporated into the UCD process in the form of short, frequent iterations and user evaluations.

Requirement fixing is discouraged in both philosophies. Customers (users in UCD terms) are encouraged to collaborate with the development team. During this collaboration, users sometimes realize that what they termed as necessary in the system are not too necessary and vice versa. At this point, the development team adjusts the requirements and other plans to accommodate these changes. Change in environment can also sometimes make change necessary.

Both philosophies stress customer satisfaction and have a people-first orientation for software development. This causes their corresponding methods to have tendency to come together and provide efficient methods for software development.

Another aspect that is common in both philosophies is the iterative approach of lifecycle. Due to the complexity of human behavior, it is impossible in UCD to create a design that is correct in the first attempt. In agile, similarly, iterations are a way to manage changes and refining the product.

We can summarize the above discussion in a table (table 2.8). It juxtaposes the two philosophies in terms how one aspect in UCD is considered in agile philosophy.

Agile Software Development	User-Centered Design
Customer collaboration	User involvement
Stakeholder satisfaction	End-user satisfaction
Developer as focus in process	End-users are focus in process, not developers
Documentation as late as possible	Documentation after every major phase
Quick delivery of working software	Frequent evaluation of prototypes
Process should be flexible enough to accommodate different projects	Process should be tailored for different organizations
Change should be reflected in next delivery	Change should be accommodated in next design iteration
Choice of which task to perform first	Choice of which technique to perform tasks

Table 2.8 Summary of Commonalities and Differences in agile and UCD philosophies

2.5 Brief Introduction to the Treatment of Requirements in UCD and Agile Methods

In this section, we will take a brief look at how the “requirements engineering” is treated in the two philosophies of UCD and agile. These two philosophies are the basis of our roadmap and play an important role in our understanding of the problem.

2.5.1 Requirements Engineering in User-Centered Design

Requirements engineering is an important phase in User-Centered Design. Unless it is clear what users want, it is impossible to make a system that can satisfy users. Requirements engineering, by definition, is a software engineering process with goals to identify, analyze, document and validate the requirements of a system (Paetsch et al., 2003).

Requirement engineering is often divided in several phases. Each of these phases plays a role in building up requirements which are vague in the beginning. These phases are usually characterized as Elicitation, Analysis, Design and Validation. According to (Cox, 2000), common phases of requirement engineering process are:

- Project Inception
- Requirement Elicitation

- Requirement Analysis
- Requirement Discovery
- Specification
- Interface Design
- Validation

Project Inception and Requirement Elicitation can be grouped in *Elicitation*, Requirement Analysis and Discovery can be grouped into *Analysis*, Specification and Interface Design can be grouped into *Design* phase, and Validation is itself a phase. Requirement Discovery is inventing new requirements from existing ones. We can validate the user requirements using the prototypes.

An important facet of requirement analysis and elicitation is context analysis. Context analysis is going to field with users and see how they use the current system. This practice gives useful insights into future system's functional and non-functional requirements. In automotive industry, for example, developing functionality from scratch is a rare practice (Weber and Weisbrod, 2003). Studying already present systems and analyzing context are crucial steps in requirement engineering. The study of user context in requirements engineering is also highlighted in process diagram by UPA (UPA, 2002).

The context in which system is to be used is identified by (Jokela et al., 2003) in terms of:

- Characteristics of intended users
- Tasks users need to perform
- Environment in which the users are to use the system.

International Standards Organization established the ISO 13407 standard for User-Centered Design process in 1999. This document is based on the definition of usability in ISO 9241-11 and tries to formulate a process that can fit into conventional software engineering processes as well.

Jokela and others (Jokela et al., 2003) discusses the ISO 13407 in detail. According to them, ISO 13407 shows limited guidance for *designing* usability. What it emphasizes is guidance for user and environment/context of use. It also has limited guidance for user goals and measures and the focus is on theoretical aspects of usability, rather than detailed coverage of methods and techniques.

ISO 13407 describes UCD from four different aspects, which are: Rationale, Principles, Planning and Activities of UCD. In *Rationale*, it explains the benefits of UCD such as reduced cost, increased satisfaction and productivity of users. *Principles* that usability is based on, according to ISO 13407 are active user involvement, appropriate allocation of functions between user and technology and multi-disciplinary design. *Planning* tries to fit the usability with the conventional software engineering process (Jokela et al., 2003). Another aspect of usability according to ISO 13407 is the *activities* of UCD. These activities include:

- Understanding the context of use
- Specifying user and organizational environment
- Producing design solution
- Evaluating design against requirements

Of these activities, the first two: *Understanding context* and *specifying user* are especially relevant to requirement engineering process. Based on this notion, we tried to incorporate these activities in our own roadmap (presented in chapter 3).

2.5.2 Requirements Engineering in Agile Methodologies

The heart of agile methodologies lies in changing requirements. The agile philosophy advocates that the requirements should never be frozen; instead, it always welcomes change and adjusts the software according to new requirements. There are different approaches to address the requirement management in different agile methods.

The traditional requirement engineering approaches and agile methods agree on the importance of stakeholder involvement. The requirements are discussed in face-to-face meetings with customers rather than through formal documents; the reason is that agile philosophy is more people-oriented than process-oriented. The customers (or customer representatives) are encouraged to be present on the development site during all phases of development. This customer is often assumed to have all the knowledge and authority in the project, which is rarely the case (Paetsch et al., 2003).

The common requirements engineering phases of elicitation, analysis and validation are present in all agile processes but with different names and do not have crisp boundaries. Techniques used are also different.

In agile methodologies, creating complete and consistent requirements documents is not considered feasible or cost effective (Paetsch et al., 2003). This makes agile methods more adaptive to change rather than being predictive of user requirements. This is considered a good quality in agile terms but certain traditional approaches discourage this idea because it makes the software development process very unpredictable.

In Extreme programming, customer reviews all the requirements and sets priority for implementation. It enables software to be developed without disruption despite of vague or constantly changing requirements. There is no artifact, however, to store requirements besides user stories. In Scrum, the requirements that are currently known are saved in Product Backlog list. The Sprints in the Scrum method involves requirement phase along with other phases (there are several sprints in a Scrum method lifecycle). In Crystal Orange, a requirements document is required; requirements to be implemented are decided before every increment starts. Feature Driven Design does not explicitly address the issue of gathering and managing requirements (Abrahamson et al., 2002).

2.6 An Overview of SUCRE Framework

The *Scenario-based User-Centered Requirements Engineering* (SUCRE) is a requirement engineering framework based on Scenarios. This framework was developed by Alsumait (Alsumait, 2004) in our research group at department of Computer Science, Concordia University. This framework is evolved from ACUDUC which in turn is derived from RESPECT framework (Figure 2.1). A brief introduction of ACUDUC and RESPECT is given in following paragraphs.

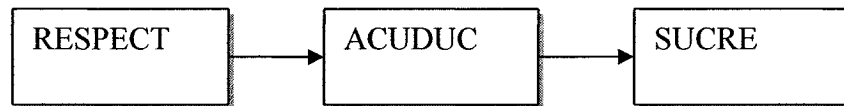


Figure 2.1: Evolution of SUCRE framework

The RESPECT (REquirements SPECification in Telematics) (Maguire, 1998) gives a framework for requirements engineering. The requirements are achieved with this framework in four phases: *Phase I – User context and early design, Phase II – Prototype and user test, and Phase III – User requirements documentation*. This framework is exceptionally good in proposing templates and forms that could be used in achieving the milestones of the framework.

The ACUDUC (Approach Centered on Usability and Driven by Use Cases) framework which combines use-cases with RESPECT, is proposed by Seffah and his team (Seffah et al., 2001). It discusses the following key activities in requirements engineering:

- Summarizing the system
- Gathering context of use.
- Functional requirement, including UI widgets
- Reviewing and Validating

These activities are defined and validated through industrial projects. Our roadmap includes these steps as its foundation.

Finally, an important work for Usability Requirements is done by Seffah and Alsumait (Alsumait et al., 2003). They have showed that Use-Case Maps (UCMs) work well for user-interface requirement engineering by extending the basic notation of UCMs and fragmenting the UCM design process into two steps, namely, the Conceptual Use-Case maps and Physical Use-Case maps. This extension of UCMs: CUCMs together with PUCMs, make up the SUCRE framework (Alsumait, 2004). This framework presents an approach for UI Requirements Engineering through Scenarios and UCMs.

The role of scenarios in requirements engineering is also studied by several researchers (Carrol, 1999, Carroll, 2000, Achour, 1998, Pohl et al., , Sutcliffe et al., 1998, Bai et al., 2002). According to them, scenarios have the potential to play important role in

requirements engineering. Some have proposed a scenario-based model e.g. (Sutcliffe and Ryan, 1998). Scenarios are beneficial in re-use of knowledge in requirements engineering because scenarios store a wealth of domain knowledge in them that can be understood by people of every level of expertise in development team and stakeholders.

2.7 Conclusions

The two prominent philosophies in software engineering that emphasize user involvement during development are agile and User-Centered Design. Four values that the agile philosophy is based on are: *Individuals and interactions over processes and tools*, *Working software over comprehensive documentation*, *Customer collaboration over contract negotiation*, *Responding to change over following a plan*. Agile methods are mainly devoted towards the implementation phase of software development lifecycle. User-centered design, on the other hand, involves users/human right from the beginning of software development lifecycle. Its methods include interacting with users frequently to get their requirements. There are several differences and commonalities in these two philosophies. Regarding treatment of requirements, UCD puts more emphasis on requirements engineering than agile philosophy. Agile philosophy believes in incorporating changing requirements during the implementation. There are several methods for User-centered requirements engineering, SUCRE is one of them. SUCRE is an evolution of ACUDUC framework which incorporates use-cases into the RESPECT framework. SUCRE framework is based on scenarios and employs use-case maps to represent these scenarios.

Chapter 3

An Agilized Roadmap for User-Centered Requirements Engineering

3.1 Introduction to Roadmap

3.1.1 Motivations for the Roadmap

As studied in previous chapter, agile philosophy has given birth to many methods for software engineering which share common characteristics. These methods are mainly concerned with implementation and are not focused on requirements engineering (Orr, 2004). Agile methods, however, handle the change during process very well besides being flexible for developers to practice. This has motivated us to combine the notion of agile philosophy with user-centered design to come up with an “*Agilized* Roadmap for User-Centered Requirements Engineering”. By ‘agilized’, we mean having characteristics of agile methods.

UCD and Agile methodologies also make good partners. Teams who have tried to incorporate these ideas have found this practice fruitful. For example, (Patton, 2002) has discovered that after having experience in agile methods and taking UCD training, when he used the mix of these methods into a team in different company, not only the team

members had no difficulty in understanding the new vocabulary, but also they readily understood who their users and their goals are.

3.1.2 Benefits of Roadmap Approach

A roadmap is a specification of the major milestones. It is an explicit specification of how to represent the objects, concepts and other process entities that are assumed to exist in user-centered requirements and the relationships that hold among them. An agilized roadmap incorporates agile aspects in the roadmap.

A roadmap is not a development tool but an approach for people who establish processes in organizations. Throughout our research in making this roadmap, we have considered the example of Daimler-Chrysler (Vehicle Company). With the description of roadmap milestones, we have presented several forms and templates that could be helpful achieving those milestones. Our roadmap, however, could be useful for any organization wishing to adopt a user-centered requirements engineering framework.

3.1.3 Roadmap versus process and methodology

The notions of *roadmap*, *process* and *methodology* are different but sometimes confusing. A *roadmap* gives major milestones that guide to the road of a higher goal: to reach the destination. It does not provide exact steps that have to be taken to reach the destination. In this chapter, we define milestones for requirement engineering for user-

interfaces and call it a roadmap. A process, on the other hand, outlines a particular way of doing something involving sequence of steps that, when performed, result in desired product or artifact. The prototype generation, as defined in chapter 5 of our thesis, is a process and not a roadmap because it mentions steps that must be taken to generate prototype.

A *methodology*, however, is a set of practical ideas and processes in a field of knowledge (Merriam-Webster, 1998). That is, a methodology is collection of processes and philosophy behind them. Agile software development is thus termed as a methodology because it has a philosophy and many methods governed by that philosophy.

3.2 The Proposed Roadmap

3.2.1 Notation used in the Roadmap

The roadmap presented in figure 3.1 is composed of milestones connected with each other. These milestones have associated with them artifacts or documents which are results of achieving those milestones. The notation used to represent the milestones, their connections and related documents is given as follows:

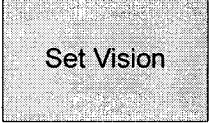
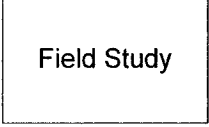
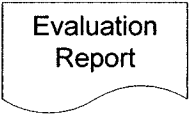
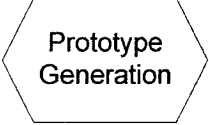
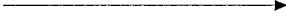

Symbols in roadmap	Meaning
	First milestone. Marks start of the roadmap.
	A milestone in roadmap.
	A document created as a result of achieving associated milestone.
	A unique milestone representing application of process to derive prototype (which is proposed in this thesis).
	An arrow marks moving to a new milestone after completing the first in normal flow of the roadmap.
	A dashed arrow marks moving to a previous milestone. It is used to represent backtracking in the roadmap.

Table 3.1: Symbols used in roadmap

3.2.2 Structure of the Roadmap

The Roadmap for UI-requirements engineering is presented in figure 3.1 below. We borrowed the aspects from agile development into this roadmap because of the belief that

agile provides the best practices for team-work and fast-delivery. On the other hand, UCD practices put rightful stress on user involvement and contextual inquiry.

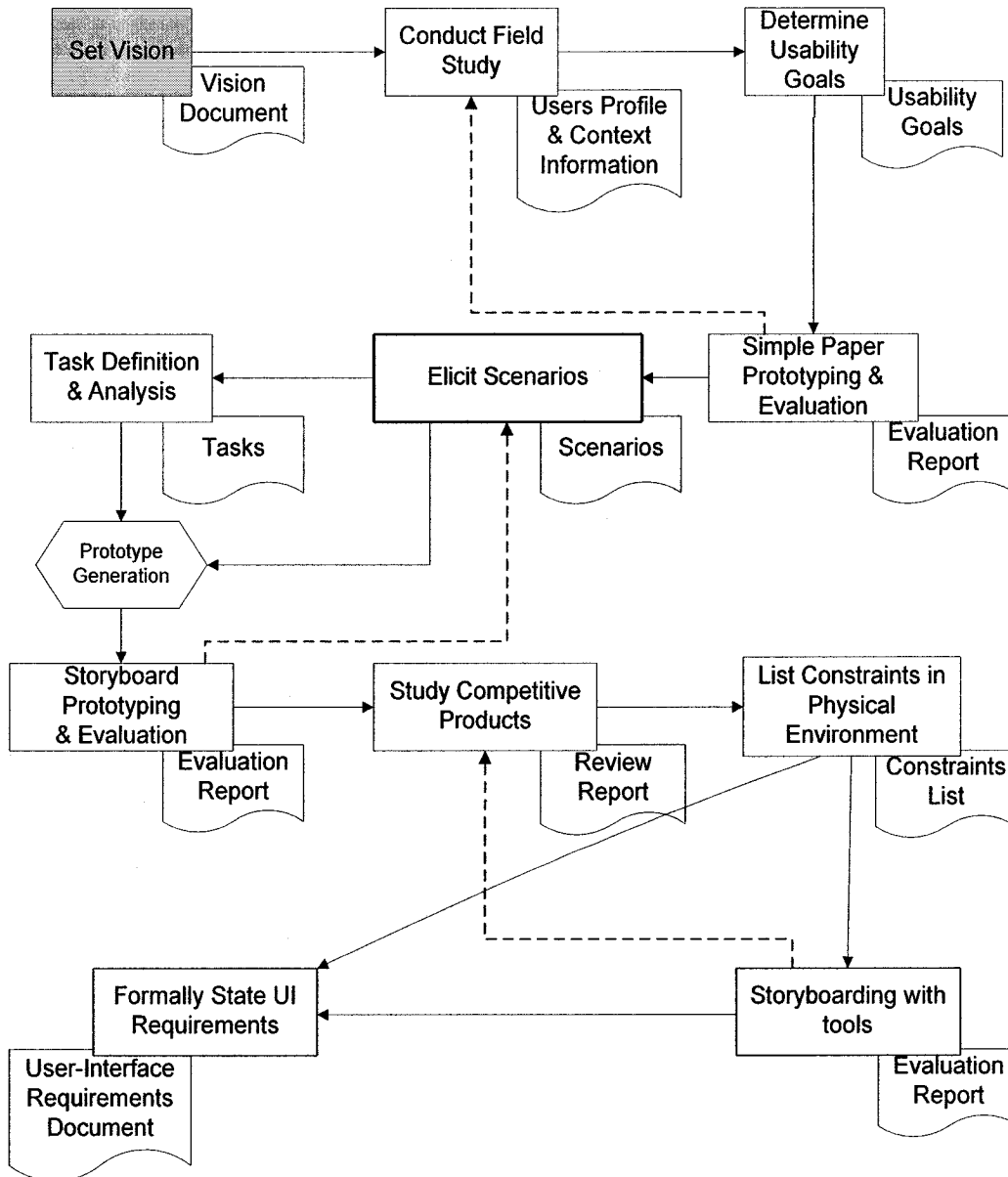


Figure 3.1: Structure of the roadmap

The roadmap is centered on Scenarios, which are supported in UCD, as well as in Agile (user stories). The scenarios are used to develop prototypes as well. The roadmap is supported by a prototype derivation process explained in chapter 5.

3.2.3 Description of milestones in the Roadmap

For the sake of clarity, we are taking here an example of requirement engineering team at Daimler-Chrysler (referred as RE-team henceforth), that is working on establishing user requirements for development of their vehicle systems. This approach will enhance the understanding and help visualizing of the explanation of the milestones. In appendix A of this thesis, we present several forms and templates derived from RESPECT framework (Maguire, 1998). These forms and templates are modified to be used by RE-team in vehicle industry.

1) Set Vision

A system's vision is an overview of a product in context of its users and environment. In other words, it gives the users' vision of the product as perceived by developers. It provides the structural blueprint for the product and how the end user will interact with and navigate through the system (Anderson et al., 2001).

In this step of Usability Requirements Engineering cycle, the RE-team will brainstorm ideas about their future product and how this product could be proved useful for its prospective users.

The output of this milestone is a *Vision Document*. A vision document includes the core requirements, key features and main constraints. (John et al., 2003)

According to (Seffah et al., 2001), a vision document should answer following questions:

1. What is the purpose of the system?
2. Why is this system necessary?

3. Who will use the system?
4. What will the users accomplish with the system?
5. Where the system be used?
6. How will users learn to use the system?
7. How will the system be installed?
8. How will the system be maintained?

In appendix A, we attach a sample vision document that could serve as a template for the RE-team.

2) Conduct Field study

This is the most important step in user-centered requirements engineering lifecycle. RE-team members will personally see people using system (e.g. driving cars) and observe their behavior while doing so. They will interview people about what they feel about using the system, what extra they want, listen to their anecdotes when they were using system alone or with other people etc.

All the information collected by this study will help RE-team to understand and classify users and observe how they use the system. Finding out scenarios of use, task definition & analysis and most of the future steps of the cycle depend on this major milestone.

This step of field study is based on the practice of Contextual Inquiry. This is discussed excellently in the book by Beyer (Beyer and Holtzblatt, 1998). The context of use description should (Jokela et al., 2003)

- (a) specify the range of intended users
- (b) be derived from suitable sources
- (c) be confirmed by users
- (d) be adequately documented
- (e) be made available to design team

During field study, some constraints in physical environment are also observed. The detailed constraints in physical environment that are to be taken care of are listed at a later stage when it is clear that which constraints to consider and which to ignore.

Appendix A contains several forms helpful in Field Study.

3) Determine Usability Goals

Usability Goals are measurable criteria to judge different usability factors of a system. These usability factors are characteristics of a system that directly or indirectly affect its users. Examples of these factors are Learnability, Memorability, error recovery & performance etc.

RE-team will focus on key features from Vision document one-by-one and determine usability goals for each feature. Sometimes, it is premature to write concrete goals but mentioning them sparingly is always helpful and it could be reviewed later on during the cycle.

According to the document *How to develop usability goals* (1996, stcsig.org, 1996),

Usability goals written in good form have three identifiable components:

Performance What should the user be able to do?

Conditions Under what conditions should the user be able to do it?

Criteria How well must it be done?

A template for writing usability goals is provided in appendix A.

4) Paper Prototyping

Paper prototyping is a simple form of prototyping where user interface is sketched on paper with minimal details. It is a low-cost and effective way to illustrate ideas and to verify them with users and colleagues. After conducting user meeting and setting usability goals, RE-team will elaborate these ideas on paper to brainstorm for future steps.

5) Elicit Scenarios

From the information collected during field study, the RE-team will extract scenarios of use for the system. A 'Scenario' is a short story (2-4 sentences) where users are sketched performing some interaction with the system. It is purely in natural language and contains little or no technical jargon. Scenarios help in extracting tasks from field study data and describing new tasks that user will perform on the target product.

A typical task that might be encountered by RE-team will look like this:

In a hot summer day, when winds are blowing like blasts of furnace, passengers ask driver to turn on the car air conditioner to cool down the atmosphere. Driver winds up all window glasses and turns on the air conditioner.

In chapter 4, we discuss a use-case maps based approach for scenario representation and elicitation.

Appendix A includes a guideline to writing well-formatted scenarios.

6) Prototype Generation

In Chapter 5 of this thesis, we provide a process to transform scenarios collected in previous step into paper prototype based storyboard. The process is based on Use-Case Maps based representation of scenarios. The prototype thus created can be ranked as mid-fidelity prototype.

7) Task Definition & Analysis

Having written scenarios, it is easier to extract tasks that user will perform with the system. Task definition and analysis involves identifying tasks and breaking them down into subtasks until they are decomposed to a set of simple actions.

This practice has many benefits including predicting time to complete task and user's cognitive load etc. There are several techniques for carrying out task analysis (GOMS being popular).

8) Storyboarding

Storyboarding is more sophisticated form of paper-prototyping where the user interface is elaborated with more detail. Different states of widgets are shown and how interaction should be done (Greenberg, 1998). In chapter 6 of this thesis, we present a scenario-based approach to develop storyboards. This approach suggests designers which widgets to put on storyboard.

9) Study competing products and previous versions

This activity provides an insight into market trends and similar systems that are being used currently. It includes competing products of different companies and also past editions of products of same company.

10) List constraints in physical environment

People are affected by physical conditions around them, so does their behavior using the system. RE-team should take into consideration the environment where the system (e.g. vehicle) will be used. Is it too hot, cold or dry, humid? Will the vehicle be used on unusual terrains like mountains or desert? Contextual inquiry is an effective method for this purpose. The constraints in Physical Environment are observed in the second step of the roadmap but are listed here because by this stage, RE-team can be well aware of the constraints to take-care of and which constraints could be ignored. At this step those constraints are only listed.

11) Storyboarding using tools

Storyboard sketches could be drawn using software tools. Examples of such tools are DENIM (Landay et al., 2000) and SILK (Landay and Myers, 1995).

12) Formally state user-interface requirements

This is the last milestone in the roadmap we have proposed. RE-team combines information gathered during all activities and formally state the requirements for user interface. User-interface requirements can be stated using LOTOS or linear textual form as outlined by Alsumait (Alsumait, 2004).

3.3 Development of the roadmap

The roadmap presented in figure 3.1 is developed with iterative approach and comes to the current form with gradual improvement. As discussed in section 3.4.4, the roadmap is based on proven methods and frameworks. In the development of the roadmap, we decided that the roadmap should have following milestones reached:

- 1) The idea of the system being built should be clear
- 2) The users of the system should be well-studied along with context in which users work
- 3) Users' task should be understood
- 4) Requirements should be verifiable

The points mentioned above are in line with the UCD concepts. Based on these questions, we created first version of the roadmap (figure 3.2). In this roadmap, users' requirements are recorded in scenarios. Scenarios are similar to story-cards used in Extreme Programming (the most common method of agile methodology). The first version of the roadmap is linear and lacks the change management (a characteristic of agile methods). It also has evaluations by users in the later stages which is not recommended in UCD practices. For the next version of the roadmap, following characteristics were required:

- 1) User's evaluation after major milestones
- 2) Backtracking to do perform activities in previous milestones if user is not satisfied.
- 3) The roadmap must be limited to requirement engineering phase. Design was out of scope of the roadmap.

The second version of the roadmap (figure 3.3) was developed with these characteristics in focus.

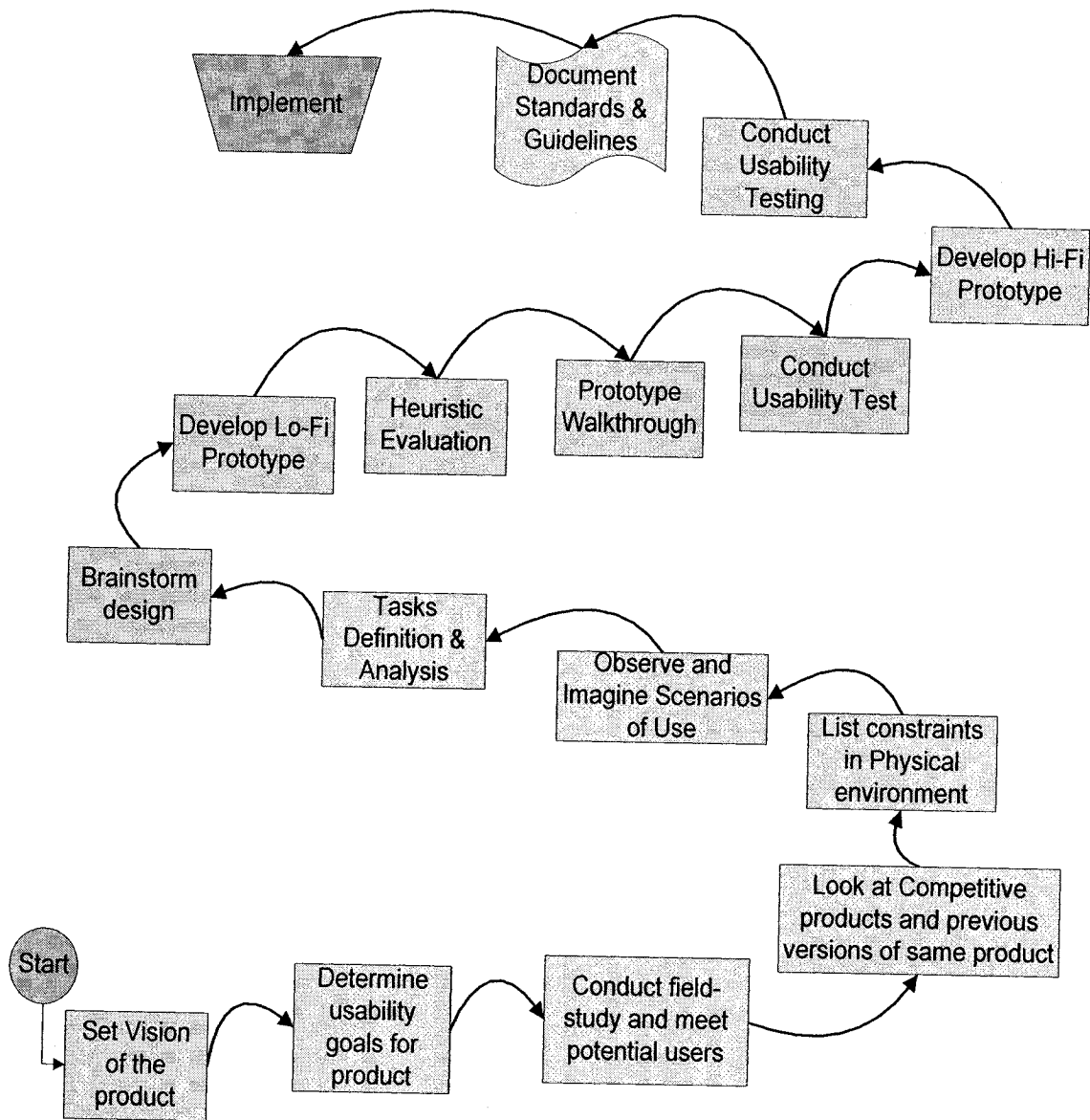


Figure 3.2: First version of the roadmap

In this version of the roadmap, users' evaluation of requirements has been made more frequent and in case of unsatisfied user, backtracking is recommended. This frequent evaluation is also advocated in agile methodology in terms of frequent releases (since this roadmap is for requirements engineering, the releases are those of requirements).

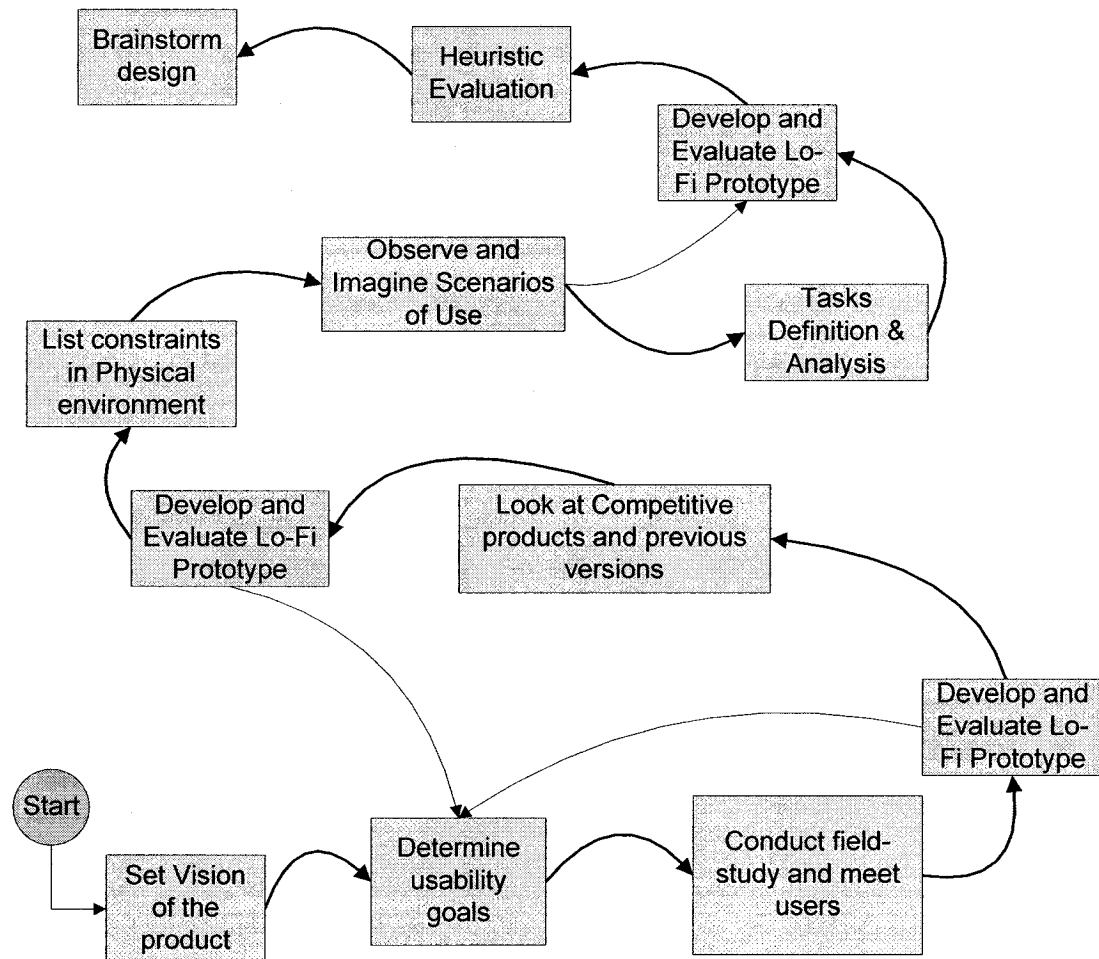


Figure 3.3: Second version of the roadmap

High-fidelity prototyping is left out since it involves extensive design decisions and the roadmap's scope is reduced to requirements engineering.

There are two points observed in this version (version 2) of the roadmap which are not fitting with UCD concepts. The usability goals are set before meeting users of the system, and secondly, scenario elicitation and task analysis is not done right after meeting users. Scenarios, as discussed in section 3.4.3, are examples of user's interaction with the system, therefore should be created soon after meeting users. Tasks are identified from scenarios as well. Agile methodology also highlights the importance of deliverables; in the next version of the roadmap, we propose two major deliverables created with first and last milestones; other documents and templates are also attached to milestones which support information exchange between the team and which help moving to next milestone. Therefore we decided following changes for the next version of the roadmap:

- 1) Field study (meeting users and observing context of use) should be conducted before setting usability goals
- 2) Scenario elicitation and task analysis should be done soon after primary information gathered in field study is verified with users with the help of low-fidelity prototyping
- 3) Documents that should be created with achievement of milestones should be shown next to the milestones in the roadmap

Third version of the roadmap is presented in figure 3.4. The milestones are represented as rectangles and each has document associated with it. The rectangle in grey shows the beginning of the roadmap (first milestone).

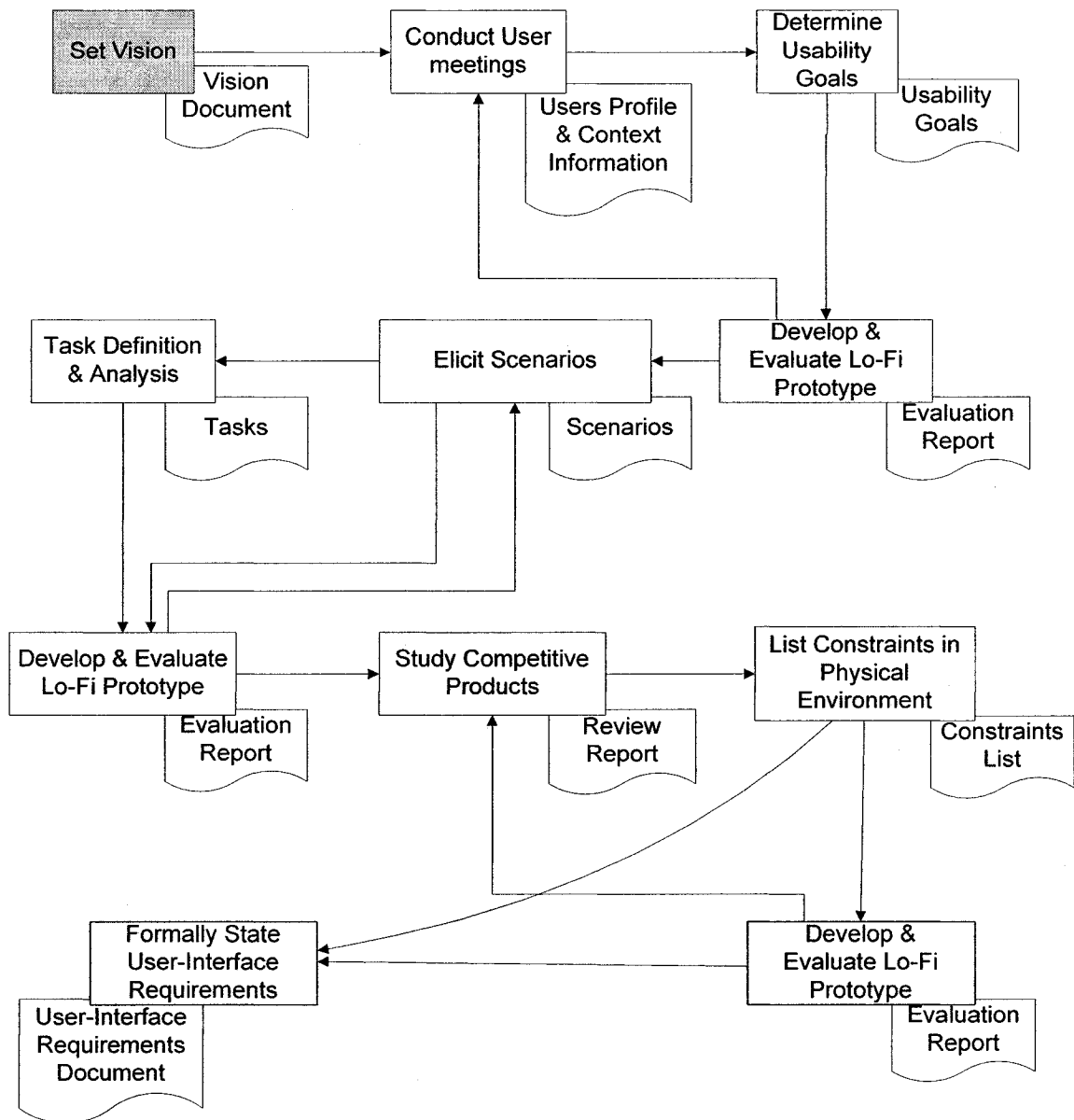


Figure 3.4: Third version of the roadmap

For the fourth version of the roadmap, we focussed on “Elicit Scenarios” milestone and devised a process to derive storyboard prototypes (chapter 5). We included that process as one of the milestones in the roadmap and decided to do mid-fidelity prototyping (storyboard is an example) in second and third prototyping milestone. Fourth and final version of the roadmap is presented in figure 3.1 in section 3.2.2.

3.4 Discussions about the roadmap

3.4.1 When to Use the roadmap?

This roadmap focuses on the requirements specification phase of the software development process. It is recommended that this roadmap to be followed alongside the normal development process and should be started with the start of the project. The key point is to begin collecting Usability Requirements right from the inception phase of software lifecycle and continue considering user as the impetus of the software till the end of the cycle. Anderson (Anderson et al., 2001) discusses the integration of usability techniques into software development in detail.

3.4.2 Agile Aspects in the Roadmap

As we discussed in previous chapter, there are several characteristics common between agile and UCD philosophies. Our roadmap reflects those similarities and shows how the milestones involved confirm to the agile concepts and practices.

Extreme programming (XP in short) uses *story-cards* for elicitation (Abrahamson et al., 2002). This is supported by our scenario-based approach where scenarios themselves are short stories. The scenarios can be written on story cards before analysis and shared by everyone. Similarly, XP is based on *small, frequent releases*. In our roadmap, whose scope is just requirements engineering, supports small releases *of requirements* (in terms of prototypes to evaluate). Other common ideas are *face-to-face communication* and *on-*

site customer which are supported by the steps of Field study, frequent prototype evaluations in roadmap (Paetsch et al., 2003).

Agile methods advocate *less deliverables, iterations*, and the idea of *decide as late as possible* and *deliver as fast as possible* (Abrahamson et al., 2002). The roadmap satisfies these ideas and gives ground for XP practitioners to follow a usability requirements gathering roadmap like this.

Agile Practices and Concepts	Roadmap features
Story-Cards	Scenarios: stories of usage
Frequent releases	Frequent prototyping
Face-to-face communication & Accessible customer	Vision development & Prototype evaluations
Iterations	Backtracking to previous steps for refinement
Less deliverables	<i>Vision Document</i> and <i>User-Interface Requirements Document</i> only major documents. Small evaluation reports are for internal communication.

Table 3.2: Agile aspects in roadmap

Prototyping, which occurs frequently in the roadmap, is also related with the agile methodologies. Orr (Orr, 2004) argues that agile practitioners should use prototypes as one of their design tools. He explains that prototyping improves the requirements definition process and is “the best way to get users engaged” in defining their requirements. The table 3.1 summarizes the agile concepts that are manifested in the roadmap.

3.4.3 Scenarios as the Major Artifacts of the Roadmap

The roadmap as we have seen in figure 3.1 has main artifact, namely, Scenarios. The data that is collected in field study is translated into scenarios at the “Elicit Scenarios” milestone of the roadmap. The scenarios thus produced are used primarily to make prototypes that are evaluated by users as well as developers.

Merriam-Webster dictionary defines *scenario* as “*an account or synopsis of a possible course of action or events*” (Merriam-Webster, 1982). In computer science, scenario is used with essentially the same meaning, with the addition of the fact that this *course of actions or events* is performed on a software system. It states, when user is working with the system, what will be one possible series of events initiated both from user’s and from system’s side (system’s response). All this information is helpful in requirements engineering.

Scenarios are an effective means of communication between developers and users as well as stakeholders (Sutcliffe et al., 1998). When users tell their stories in natural language, a member from development team have to listen carefully and paraphrase the user's narration into an effective scenario. Analyst then checks the final scenario with the domain expert and uses it as a communication means between the development teams. Customers also find it easy to refer to concrete scenarios rather than abstract and complex models (Pohl et al., 1998).

When designing the user-interface of system, scenarios can guide through different steps of user interaction. In words of Suttcliffe (Sutcliffe et al., 1998), "scenarios may be seen as pathways through system usage". As pathways guides where to turn and where to proceed, the scenarios guide the UI designers to visualize the user's workflow in very concrete terms and design better and effective interfaces. A scenario expresses an example of behavior of system and users (Achour, 1998).

According to Pohl (Pohl et al., 1998), computer industry is interested in using scenarios and use cases. There are several management benefits in using scenarios as well. Scenarios help in division of labor due to their concrete nature in stating the functional requirements. Knowledge re-use in scenarios is beneficial in requirements engineering, but reusing scenario-based knowledge is somewhat difficult because scenarios are instances i.e. specific examples of system usage. Another characteristic of scenarios is that they are complex artifacts that need to be managed. (Sutcliffe et al., 1998)

Scenario-based approach could be used at every phase of system development with certain modifications. The scenario-based approach bunch mainly in four phases of: requirement elicitation, requirement analysis, human-machine interface design and validation (Cox, 2000). Majority of these phases are related to the field of requirements engineering which is also the focus of our thesis.

A typical scenario-based approach to requirements engineering involves elicitation, specification and analysis. Cox (Cox, 2000) maintains that scenarios can be made fit to any requirements engineering model. There are, however, some steps that must be taken to fit scenarios into requirements process.

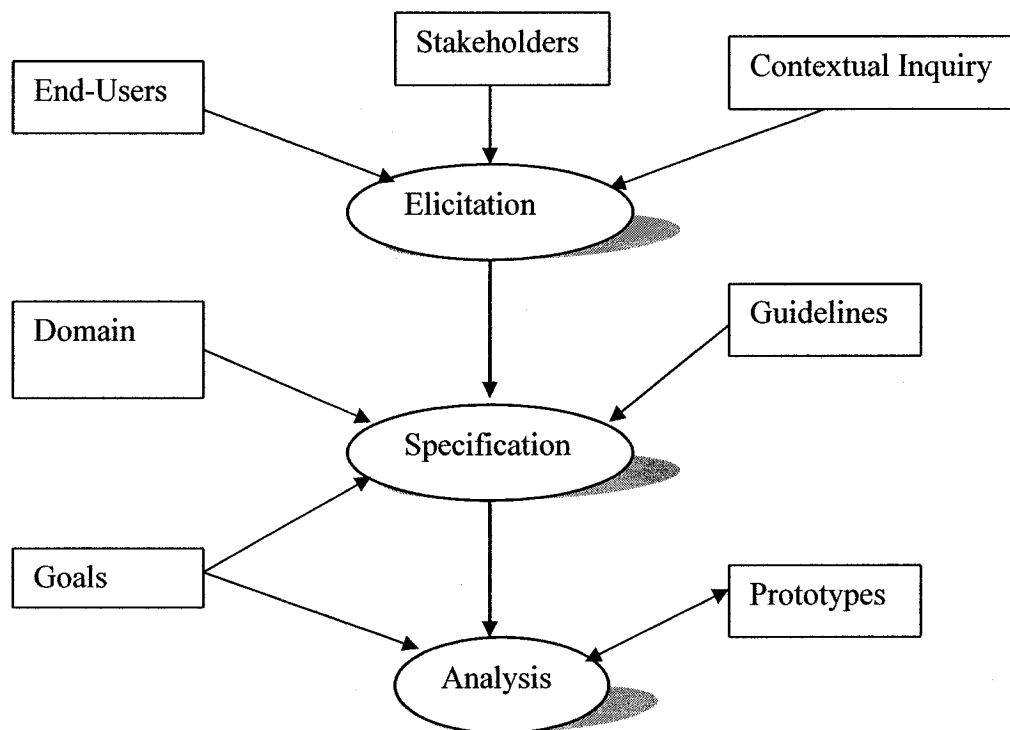


Figure 3.5: Typical steps in scenario creation

As seen in the figure 3.5, eliciting scenarios involves contextual inquiry, which is a vital practice in user-centered design, we can relate how scenario-based design in requirements engineering can overlap the User-Centered Design process. Specification of scenarios is a knowledge-extensive job and needs input from domain experts (Pohl et al., 1998). Attached to every scenario is a goal (Achour, 1998) and must be kept in front when writing scenarios. Achour (Achour, 1998) gives important guidelines on writing textual scenarios. Carroll (Carroll, 1999, Carrol, 1999) points out that there are some characteristic elements of a scenario: *a setting, agents or actors, a plot;*. (typical elements of every story). Existence of prototypes is important for scenario analysis and validation. Pohl (Pohl et al., 1998) argues that “without prototyping, the value of scenarios would drop almost to zero, and vice versa” and that “scenarios and prototypes complement each other in a symbiotic manner”. In our thesis, we propose a connection between scenarios and prototypes via use case maps.

These steps on scenario creation are in line with the roadmap. Before Scenario elicitation begins, data is collected from users and their context. Specification is done with the help of domain experts and guidelines. In our roadmap, the specification is supported by SUCRE framework (Alsumait, 2004). To analyze the scenarios for requirements, the roadmap has a milestone of Prototype Generation. The process for prototype derivation is discussed in greater detail in chapter 5. Scenarios thus play the pivotal role in the roadmap.

3.4.4 Basis and Validity of the Roadmap

The roadmap presented in following section is based on the study of several proven methods and frameworks which integrate Usability into software engineering processes. These works deal with different aspects and levels of integration, for example, functional requirements, validation etc.

Anderson (Anderson et al., 2001) explains a process of putting User-Centered Design practices in Rational Unified Process (RUP). Figure 3.6 summarizes the steps of requirements definition during software development. These steps provide a ground for our roadmap. We defined our roadmap with same logical ordering of steps as in figure 3.6; i.e. Creating a vision of the project, conducting field study and user modeling, analyzing workflow (scenarios and task modeling) and making a product vision in terms of prototypes. Anderson and others (Anderson et al., 2001) used this idea in integration of User-Centered Design practices into RUP and tested on real projects.

The ACUDUC (Seffah et al., 2001) and RESPECT (Maguire, 1998) frameworks discussed in section 2.6 also provide the basis of the roadmap. Another major inspiration for the roadmap was the process for “designing the user experience” defined by Usability Professionals Association (UPA, 2002).

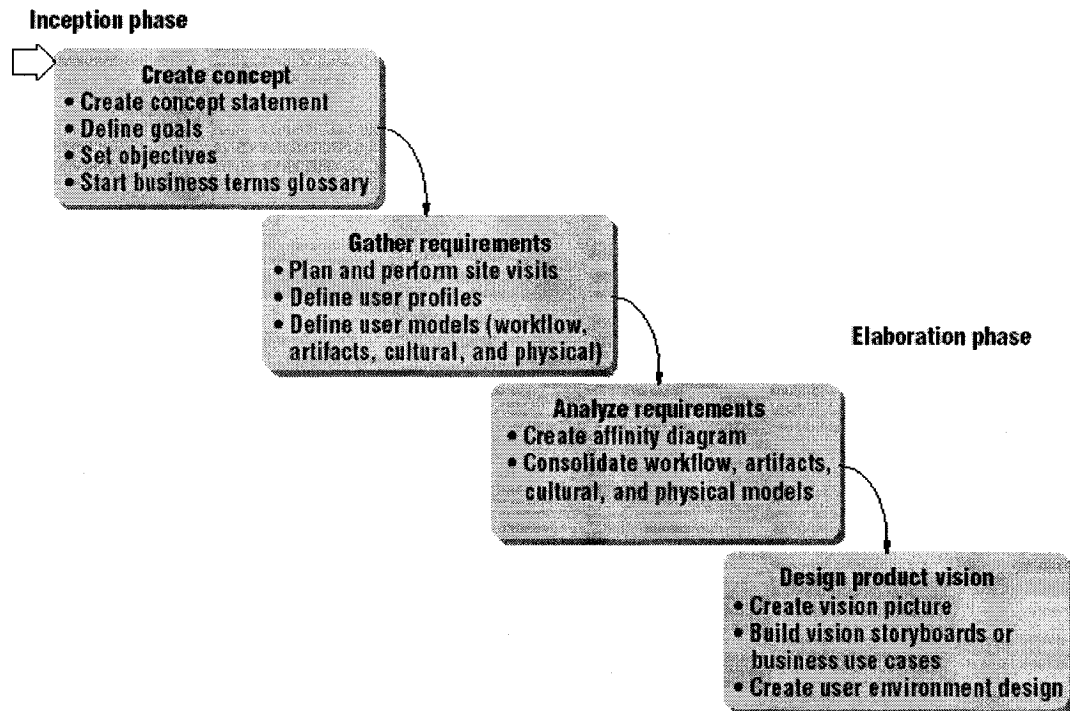


Figure 3.6: Integration of UCD practices in RUP (Anderson et al.)

Besides having all these frameworks and methodologies, there is still a need for a roadmap that unifies all the ideas and complements it with even more fresh concepts like that of agile methodology. The roadmap also supports the Scenario-based and User-Centered Requirements Engineering (SUCRE) framework by Alsumait (Alsumait, 2004). This roadmap was primarily prepared with needs of Daimler-Chrysler (Vehicle Company) in mind. In future, we recommend the testing of this roadmap in Daimler-Chrysler environment. We have helpful forms and templates for several steps of roadmap adopted from RESPECT framework for a vehicle industry. This will support the future validation of roadmap in Daimler-Chrysler Company.

3.5 Conclusions

Agile philosophy has many methods associated with it. These methods primarily address the implementation phase and do not have requirement methods especially for user-interfaces. The user-centered design, however, can benefit from many ideas from agile methods. An important contribution for user-centered requirements engineering is to make a user-interface requirements roadmap that incorporates useful concepts of agile philosophy with user-centered design methods. The roadmap presented in this chapter helps in requirements gathering for user-interfaces and has agile aspects in it. It is based on the SUCRE framework for scenario-based requirements engineering. The main artifact of the roadmap is scenario which is supported with a scenario specification process in next chapter.

Chapter 4

Scenarios and their Representation as Use-Case Maps

4.1 Introduction

In the previous chapter we presented a roadmap for usability requirements engineering. That roadmap is driven by a major artifact, namely, Scenario. Here, we will discuss in detail how the scenarios could be represented.

Representing scenarios is important for their deeper analysis. There should be three characteristics in a representation in order to be deemed good. Firstly, the notation should be easy to understand by developers as well as users/stakeholders. Secondly, it should enable the requirements to be validated through scenarios. Thirdly, it should be easier to derive prototypes from scenario representation. We found that Use-Case Maps have all these characteristics.

Use-Case Maps (or UCMs in short) were proposed by Buhr (Buhr, 1998) as a means of representing scenarios in understandable and standardized form. These UCMs are proved

helpful in explaining behavior of many different types of systems including real-time, distributed as well as user-interfaces of systems.

User-interfaces are the most important part of software from user's perspective. Use-Case Maps as we will see in section 4.2 are an important way to represent scenarios. However, to extract requirements that are particular to user-interface, there need to be provision in UCMs for capturing those requirements. The UI extension to UCMs (in SUCRE framework) caters this need.

In this chapter, we will discuss the representation of Scenarios as Use-Case Maps in detail. Later on, we will explore how UI extension of UCMs in SUCRE framework enables to get user-interface related requirements.

4.2 Use-Case Maps as Scenario Representation

Use-Case Maps were first developed by R.J.A. Buhr of Carleton University Ottawa, and its official supporting website started in 1998 (Amyot, 1998). Since that time, UCMs are used in various software engineering fields.

Use-Case Maps represent causal relationships between *responsibilities* which may be bound to different *components*. These components could be software (plug-ins, classes, databases) or non-software (actors, user, etc) as well as of user-interface (Amyot, 2000). In our context, the components are user-interface related and users who use that user-interface.

To elaborate, responsibilities are actions that user performs. Responsibilities could represent ‘tasks’, in a broad sense, that could be composed of several smaller actions (e.g. the task of entering PIN is composed of several actions of keystrokes on keypad). These responsibilities are building blocks of user’s interaction with the system. User performs these responsibilities in a particular sequence and the system responds to these actions. The end-result is a completed task. The sequence in which these responsibilities are performed is also important. If this sequence is changed, the overall result of the scenario will be changed. Also important are the components on which actions are done.

A UCM is an apt tool for communicating all the aspects of a scenario. The structure of UCMs allows the representation of all the aspects of a scenario, namely, the actors, the flow of actions and its start/conclusion. With the layering of UCMs, complex scenarios can be broken down into sub-scenarios (containing sub-tasks) which help in simplification for understanding. This portrayal of scenario helps in achieving further goals related to usability requirements engineering tasks including prototyping.

4.2.1 The Basic Notation

The basic notation of UCMs is simple. Components are represented with rectangles. The flow of scenario is represented with lines that pass through component rectangles. The responsibilities that must be performed on the components are represented with labeled crosses over the line. Figure 4.1 shows a simple use-case map for logging-in to ATM.

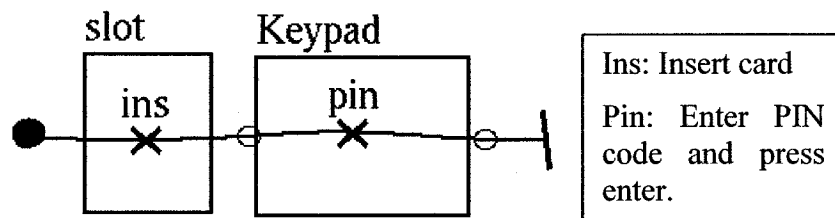


Figure 4.1: Simple UCM for ATM login

If several components are part of a bigger component, the small components could be grouped together in a larger rectangle (figure 4.2). This is called decomposition in UCM terms (Buhr, 1998). One kind of UCMs is Unbound UCMs which are drawn informally at very early stages of analysis (Figure 4.3). The characteristic of Unbound UCMs is that no

components are shown, just the line and crosses show the sequence of actions involved in scenario (Buhr, 1998). If a scenario is complex and contain a sub-task, a UCM can be layered. A stub could be used to simplify the main UCM and showing the sub-task as a plug-in. This plug-in is shown as a diamond in UCM notation.

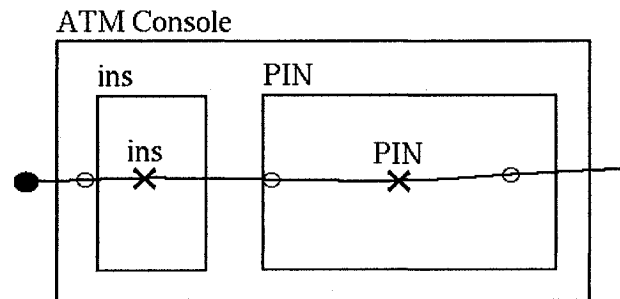


Figure 4.2: Decomposition of ATM Console into two components

With the help of this notation, one can express from the simplest to the most complex scenarios. Unbound UCMs are great for preliminary brainstorming when components of a system are not known, or are not relevant. We can keep the scenario flow and responsibilities intact while changing different components to reuse the UCM for different system environments.

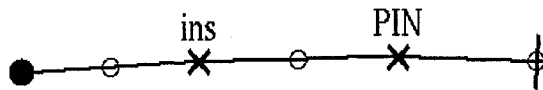


Figure 4.3: An Unbound UCM

4.2.2 Advantages of UCMs

Use-Case Maps is a rich notation to represent use-cases and scenarios. It is not a behavior specification technique, as noted by its inventor Buhr (Buhr, 1998), but a way to easily show the flow of scenarios through lines and boxes.

UCMs help visualizing the behavior of a system, discussing it with domain experts, sharing with developing team, and brainstorming the future design of the system. It helps to see the big picture of the system rather than focusing on details. Due to its simple notation, it is easily understandable by developers as well as the technical and non-technical stakeholders.

The composition of UCMs holds the scenario path as a part of representation. The lines that show the path passes through the components but is still independent of them (Buhr, 1998). That is, the responsibilities across the path of the scenario are performed on different components of the systems, but in case of different architectural entities, these responsibilities would be performed on other components. This makes the scenario path independent of system components (Amyot, 2000). An example of this is a UCMs of personal-identification-number (PIN) verification in *ATM* and *Online banking website* (Figure 4.4). Although the path and responsibilities involved are same, but components that do the actual verification could be different in case of ATM or Online Banking web-application.

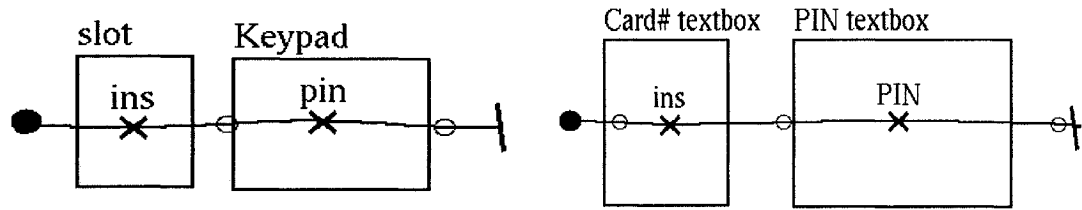


Figure 4.4: Simple UCMs for ATM and Web-Application login

Being a visual notation, UCMs show paths of scenarios that cross through components. If a scenario contains alternate paths, the paths are also shown with the help of AND/OR forks and joins. This notation helps in observing a behavior pattern of a scenario at an early stage of analysis. This could lead to pruning of inefficient or unwanted paths of interactions resulting in a better design.

This novel notation is useful for capturing functional requirements (Amyot, 2000). Buhr maintains that this notation is intended to be useful in requirements specification, design, testing, maintenance, adaptation, and evolution (Buhr, 1998). Recent studies show their usefulness in non-functional requirements as well (Alsumait et al., 2003).

Use Case Maps are now subject of many research groups. Research is being done for using UCMs in reactive and distributed systems, telecommunication, graphical user-interface, banking and business, agents, web-applications and networking (Amyot, 2000). Several studies try to combine UCMs with UML and denoting UCMs in XML form (Alsumait, 2004). Miga tries to derive Message Sequence Charts (MSCs) from UCMs (Miga et al., 2001). In their paper, Seffah and team (Seffah et al., 2003) present a UI-User Requirements (UIUR) model which includes UCMs at its core.

4.3 Use-Case Maps in SUCRE framework

UCMs have proved to be very effective in communicating scenarios. Many people are doing research to adapt UCMs in different fields and it is currently applicable to a wide range of areas. It is also a topic of several research projects (Amyot, 2000, Amyot, 1998).

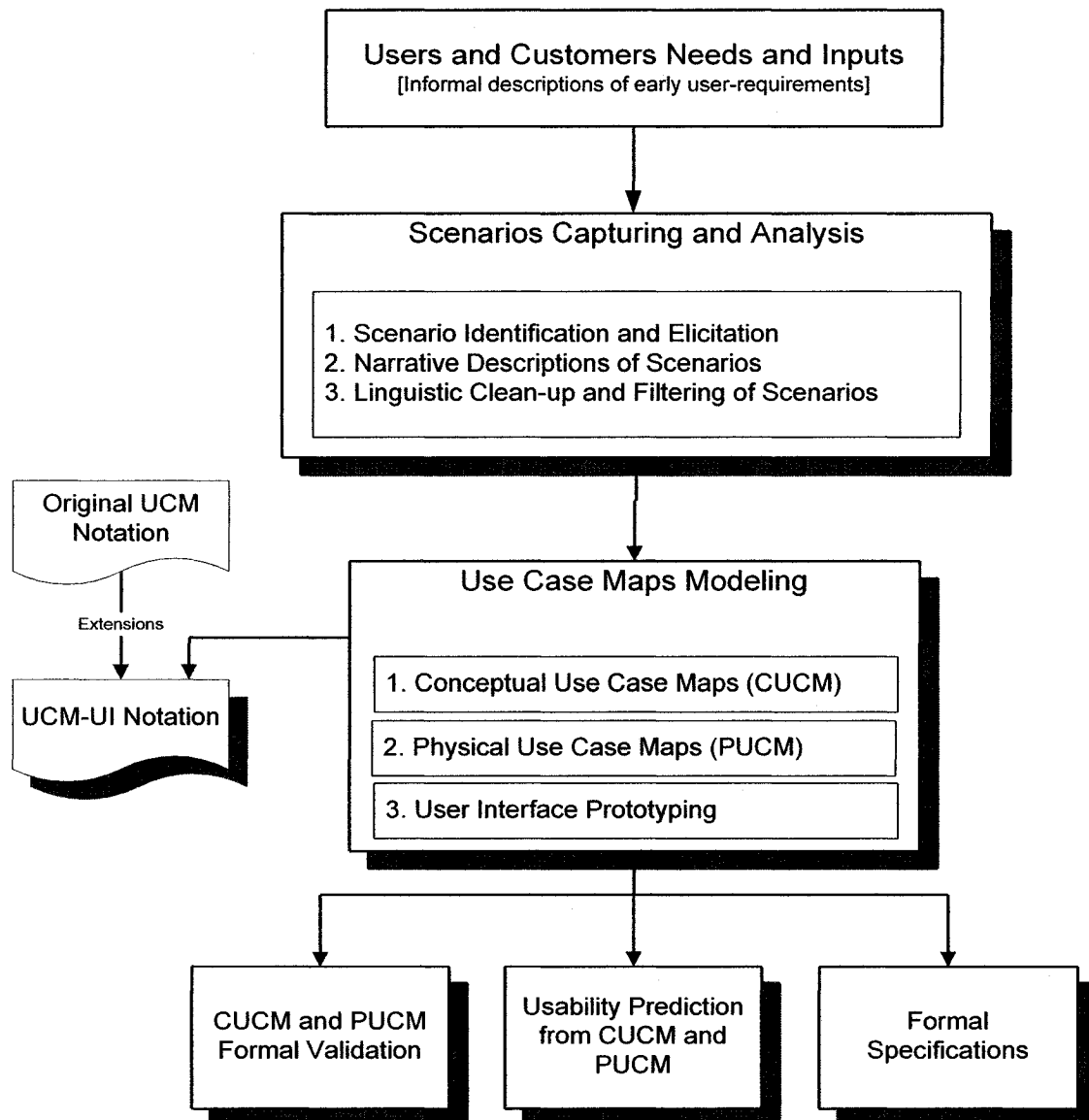


Figure 4.5: SUCRE framework (Alsumait, 2004)

As introduced briefly in section 2.6, an important work for UCMs in the field of user-interface design is done by Seffah, Radhakrishnan and Alsumait (Alsumait et al., 2003). They have extended the original UCMs so that there are now two tiers of use-case maps, namely, Conceptual and Physical (CUCMs and PUCMs respectively). Figure 4.5 gives structure of the SUCRE framework

The Conceptual Use-Case Maps (or CUCMs) capture the scenario in abstract way. They just outline the main tasks that are to be performed and the system's dialog with the user (system asking user some question and user responding to it). Since basic UCM notation

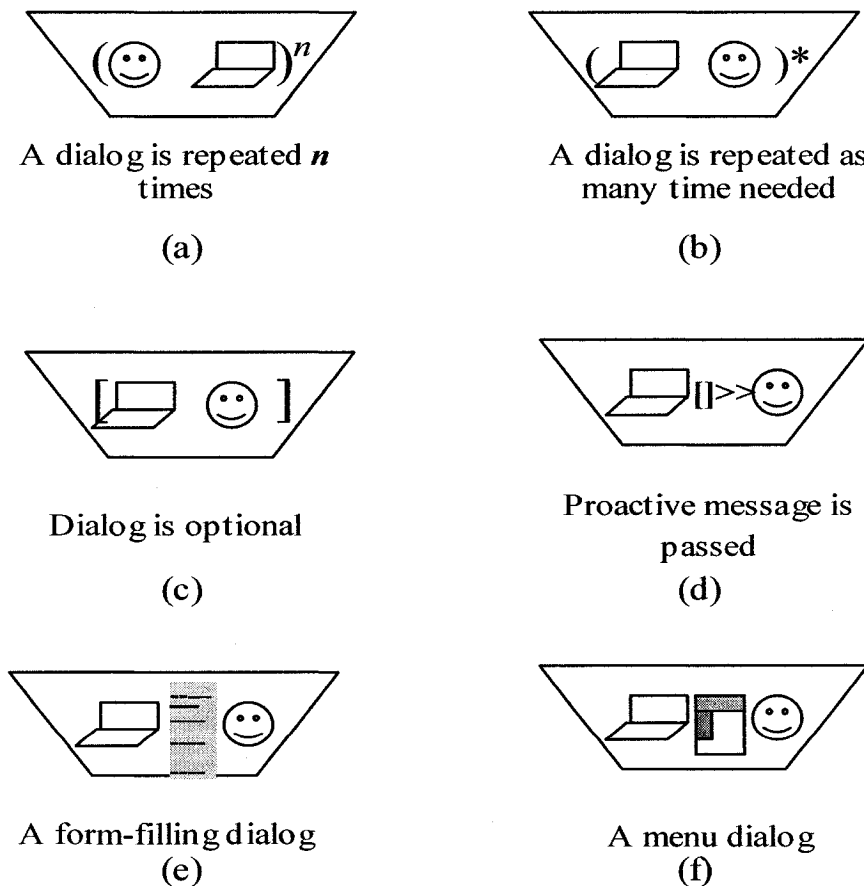


Figure 4.6: Dialog symbols for CUCMs (Alsumait, 2004)

lacks system-user dialog notation, they have added a new set of symbols to represent different types of dialogs (Alsumait et al., 2003). Figure 4.6 shows dialog notation used in CUCMs

The second layer of UCMs introduced by Asmaa (Alsumait et al., 2003) is Physical Use-Case Maps (or PUCMs). These PUCMs elaborate the scenario over symbols of parts of real system i.e. the PUCMs contain Presentation Units (PUs in short). These PUs are rectangles with icons that represent some real component of systems (e.g. menus, buttons, etc). The path of scenario is still shown in wiggly lines with crosses representing responsibilities, but here, these crosses appear inside PUs denoting that this responsibility has to be performed in a part of UI that is denoted by this PU. There is one important point that we want to make here: PUCMs are not equivalent or substitute for Prototypes. The prototypes are designer's proposal of user interface. It includes real-like elements of original interface and depending on its level of detail, user can interact with the prototype. PUCMs however show only the flow of this interaction.

The PUs suggested by Alsumait (Alsumait et al., 2003) to represent some parts of presentation are listed in Figure 4.7.

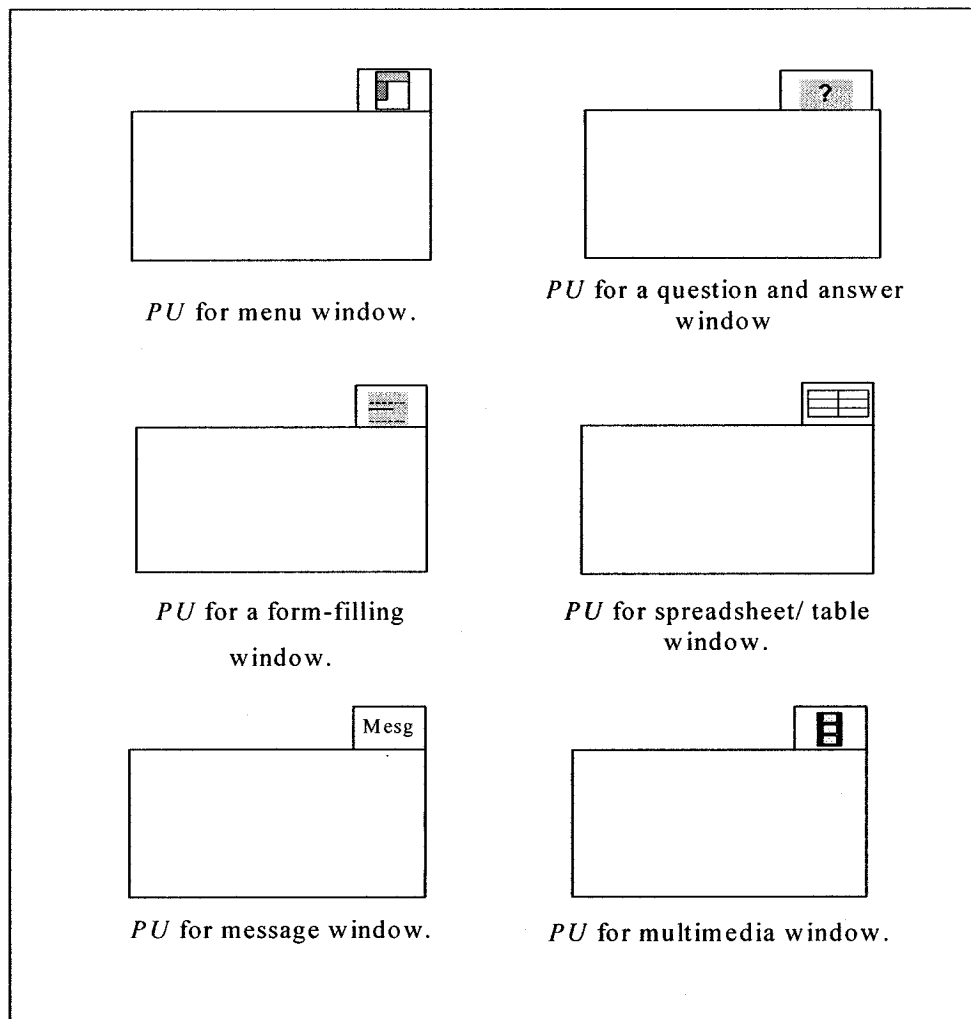


Figure 4.7: Presentation Units (PUs) suggested by (Alsumait et al., 2003)

This extension of UCMs, CUCMs together with PUCMs, make up the SUCRE framework (Alsumait, 2004). This framework presents an approach for UI Requirements Engineering through Scenarios and UCMs. In chapter 6, we present an example of representing a scenario in UCM notation.

4.3.1 Advantages of SUCRE framework

The advantages of using SUCRE framework are numerous in user-interface development. User-interfaces are complex entities. Behind their physical/tangible components, a conceptual model plays an important role. Placing components on an interface without having a clear conceptual model may result in inefficient interface. The conceptual model helps designers first understand the purpose of the interface. This includes the tasks to be performed and other constraints.

Once the conceptual model is made, it could be verified with usability engineers and users. After finalizing the conceptual model, a physical model helps the designer to decide which groups of elements could be placed on an interface and in what order. This is important as a middle layer from a conceptual model to a prototype. In the physical model, the same flow of scenario path through components is stated and verified.

SUCRE framework's two-tiered approach is meant to accomplish the above-mentioned advantages of conceptual and physical modeling. The CUCMs show the conceptual model behind a user-interface. This conceptual model is then translated into a physical model through PUCMs. The notation for both CUCM and PUCM enable designer to express their ideas in flexible way, while keeping in mind the basic UCM notation on which CUCM and PUCM are based.

CUCMs and PUCMs give a great deal of flexibility to design user-interfaces. We strongly believe that this notation can be further enriched to encompass all aspects of user-interfaces. In the next chapter, we propose enrichment to this notation and propose a way to formulate prototypes through this two-tiered UCM technique.

4.4 Conclusions

Scenarios are the main artifacts created in the roadmap discussed in chapter 2. After the creation of scenarios, the next step is to represent them in such a way that further processing could be done on them and, where possible, the processing could be automated. This need was fulfilled well by Use-Case Maps (UCMs) by Buhr (Buhr, 1998). The reason for choosing UCMs for representation of scenario is the presence of three characteristics in UCMs: the notation is easy to understand by developers as well as users/stakeholders, secondly, it enables the requirements to be validated, and thirdly, it is easier to derive prototypes from scenario representation. The basic notation of UCMs is extended by Alsumait (Alsumait, 2004) in SUCRE framework to enable the UCMs to represent user-interface-related scenarios. With the new extension of UCMs, there are now two tiers of Use-Case Maps, namely, Conceptual and Physical (CUCMs and PUCMs respectively). Two new sets of notation are proposed in the SUCRE framework to draw CUCMs and PUCMs. In this thesis, we extend the PUCM notation set to include more types of presentation units in user-interfaces. This extension is proposed in next chapter.

Chapter 5

Prototype Derivation from Scenarios

5.1 Motivations for a Prototype Derivation Process

Prototypes are great tools to expound users' requirements and to verify analysts' understanding of user-needs. These are a means of showing an idea behind a design in an inexpensive way (Z.Guan and Luqi, 2003). They give an early flavor of product and what it will be like. At requirements gathering phase, when user-centered design team has just conducted field study and collected usage-stories, scenarios are made that summarize and filter all usage stories; prototypes are then made that contain designer's proposed interface and how user will be using the system. Prototypes are also helpful for a more formal requirement that must describe user interface if a subcontractor is to deliver a system (Lauesen, 1997).

There are some limitations in conventional prototyping. Paper-prototyping is an intuitive process and depends completely on designer's imagination and knowledge of user behavior. Prototype designers are well-versed in the field of design and representation but not always so in usability requirements engineering. If the designers are to build prototype from requirements, it is possible that they will build nice representation but that might not necessarily reflect the requirements. Another limitation in conventional

prototyping is lack of standardization. A paper prototype can very well fill the space with different kinds of symbols or pictures, but when it comes to understanding those symbols, different people (including end-users and developers) can understand different meanings. This is due to the fact that the symbols used in prototype did not follow a set standard. A designer's perception of a simple menu could be understood as a push-button by user.

To help designers decide which elements to put in paper prototypes, we propose in this paper a process based on PUCMs that are derived from scenarios (PUCMs are discussed in chapter 4 in detail). The Presentation Units (PUs), that are building blocks of PUCMs, are abstract symbols that are generic for a larger range of user-interface components.

To make a prototype that really reflects requirements, it is better to completely automate the prototype generation. It involves a software tool that will let the designer draw conceptual and physical use-case maps and that tool will intelligently read the use-case map and generate a prototype. This kind of software requires extensive programming besides the research and is not feasible due to limited time available. Instead, we provide a tool in the form of a matrix (or table) that can suggest the parts of prototype based on the physical use-case maps.

5.2 Prototyping in the Agilized Roadmap

The agilized roadmap presented in chapter 3 (figure 3.1) for usability requirements engineering has important milestones related to prototyping. Prototyping is done after every major acquisitions of information from user or their context to verify that information. This is especially important after the scenario elicitation and task analysis where major features in the user-interface have to be explored. At this stage, a mid-fidelity prototype in the form of storyboard is helpful in visualizing the flow of future interface.

The prototype derivation process discussed in this chapter is meant to help the designers decide which elements to draw on the prototype. The milestone in roadmap of scenario elicitation (*Elicit Scenarios*) creates detailed scenarios and their representation in use-case map notation. These scenarios are input for the prototype derivation process which results in storyboards. Storyboards can be discussed with users and stakeholders, as well as in development team.

5.3 Prototyping in SUCRE Framework

SUCRE was developed by Alsumait (Alsumait, 2004) as a scenario-based framework to support usability requirements engineering. It is based on Use-Case Maps as a tool for representing scenarios.

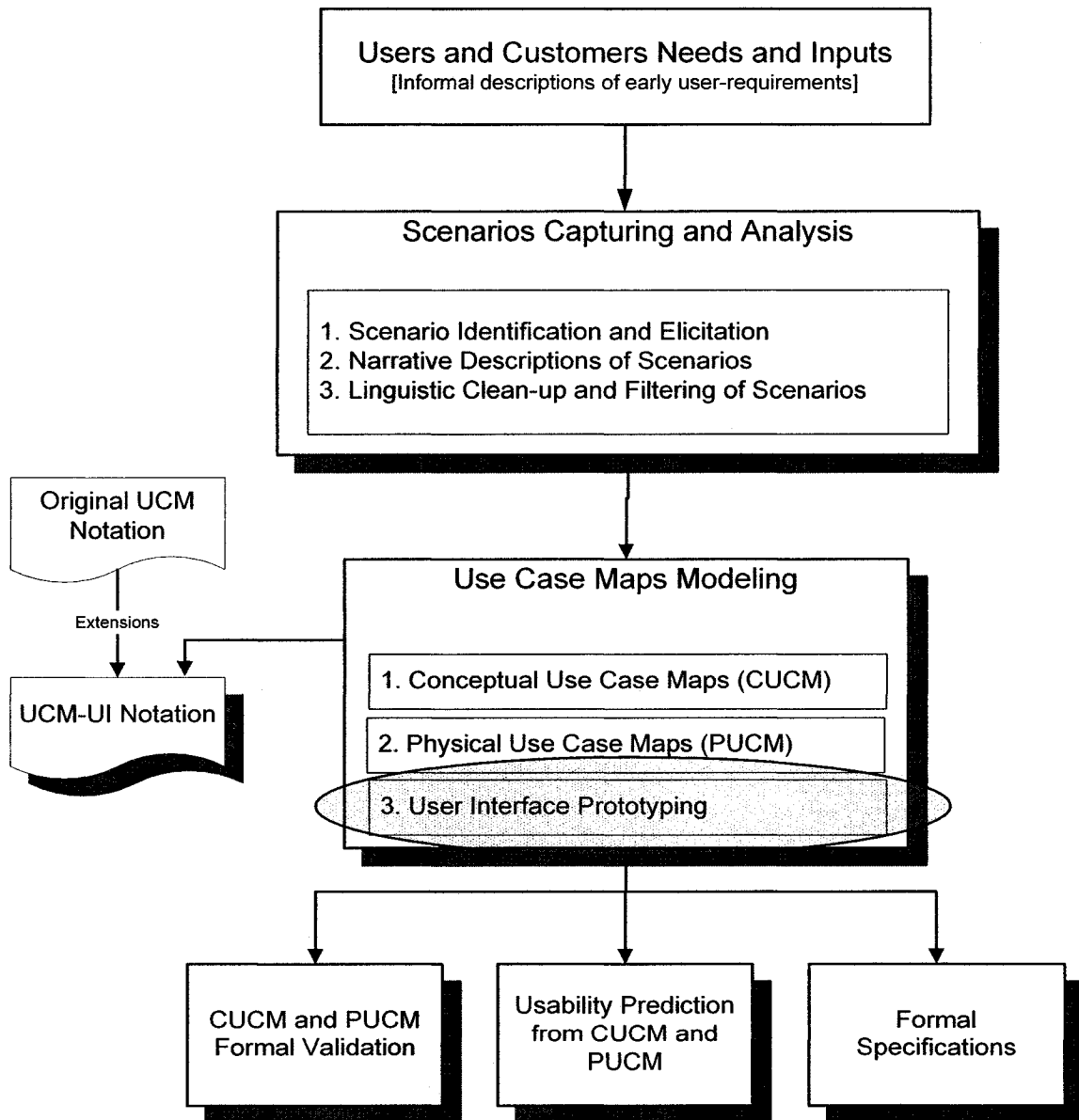


Figure 5.1: Prototyping in SUCRE framework (Alsumait, 2004). The prototyping step is circled.

The framework proposes a two-tier approach to represent UI-based scenarios. These two tiers, known as CUCMs and PUCMs are explained in detail in previous chapter. After representing the scenario in UCM, the framework applies different techniques to extract and validate requirements. The derivation of prototypes from UCMs is mentioned in the framework (figure 5.1), but no process is outlined to achieve this goal.

As seen in figure 5.1, the user-interface prototyping step follows the CUCM and PUCM. This makes prototyping an important step of the framework. To complement the framework with a process that can serve in prototype derivation, we provide a way to derive prototypes from use-case maps in next section.

5.4 Process of Prototype Derivation from PUCMs

The derivation process itself is not very complex (figure 5.2). Starting from a scenario, the usability requirements engineer will make use-case maps of the scenarios created. According to the SUCRE framework (Alsumait, 2004), the UCMs are developed based on two models; Conceptual and Physical. The result is Conceptual and then Physical UCMs. Taking PUCMs as input and having user's intended actions handy, the tool for mapping PUCM parts to prototype elements (Mapping Matrix) is then employed to suggest the prototype elements from the collection (in Appendix C) and a prototype is made.

For applying the translation, the task that is supposed to be performed on prototype screen must be known. Since there could be a choice of prototyping elements for a PU, knowing an *intended action* will help decide the accurate element(s) that should go on the prototype. Being aware of intended actions on user-interface also makes sure that both the analyst and designer are conscious about the user's needs and task details; this, in turn, will ensure informed decisions about the user-interface.

Before presenting the Mapping Matrix tool, we first extend the Presentation Unit (PU) notation suggested by (Alsumait, 2004) to include most common user-tasks. We then present UI symbols to be placed in prototypes. These symbols are generic and represent the typical components of different kinds of user-interfaces.

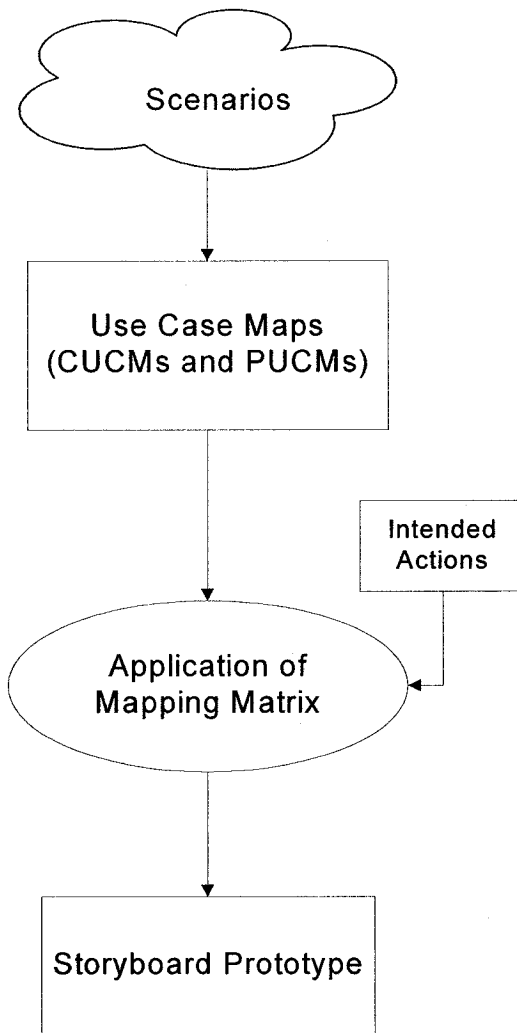


Figure 5.2: Process of Prototype Derivation from Scenarios through UCMs

5.5 Extension to Presentation Units (PUs)

The original set of PUs proposed in SUCRE framework (Alsumait, 2004) is given in figure 4.7. This notation is limited and was designed only for the purpose of explaining the framework. Here in this paper, we will try to extend this notation by designing new, more intuitive icons for already present PUs and also proposing new ones. Each PU is given a fixed name and they will be referred to by those names.

5.5.1 Rationale for extension of PU Symbols

The symbols we propose here for representing presentation units in PUCM reflect the actions that can be performed on the actual UI elements they encompass. While these PUs give a hint of actions that could be done on presentation units, their function is only to give an idea about the placement of actual user-interface element on the screen and its size proportion related to other elements.

The set of PUs proposed originally in SUCRE framework (figure 4.7) were given, as mentioned before, for explanation of the framework and were enough for that purpose. But for development of a tool for prototype derivation, we need a set of PUs that covers most of the presentation and interaction aspects in user-interfaces. Following interaction tasks or presentation parts were not supported in the original set of PUs (figure 4.7):

- 1) Viewing or editing textual, graphical or mixed contents
- 2) Indicators of system status or quick feedback
- 3) Value changing parts of a user-interface (important in vehicles)
- 4) UI parts that require constant system-user interaction
- 5) Audio/sound/voice based interaction
- 6) Interaction in the form of navigation

To support these fundamental presentation parts of user-interface, we created more presentation units and designed symbols to represent those presentation units. We included some of the original PUs without changing (e.g. Form PU) and modified some to make new PUs that could cover similar aspect of interaction (e.g. “Menu window” of original notation has been changed to “Tools collection PU” to include fixed/docking toolbars as well).

The resulting set of PUs (appendix B) covers most interaction tasks and presentation parts. In the tool for prototype derivation, each of these PUs stand for one or more PEs (Prototype Elements, discussed in section 5.6). Extension of PU collection is an important contribution to SUCRE framework.

5.5.2 Some Presentation Units for PUCMs

Following table 5.1 shows some of the PUs that we propose for PUs. The complete collection of our suggested PUs is given in appendix B.

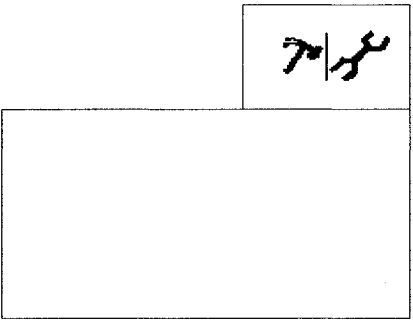
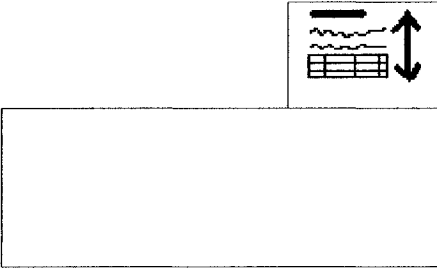
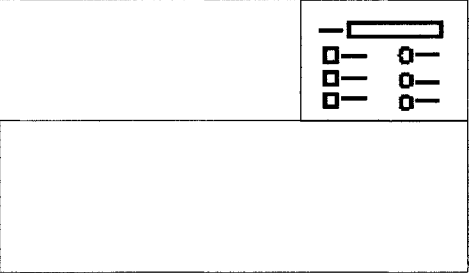
Name, Description & Example	PU Symbol
<p>1. Tools Collection PU</p> <p>Description: User can choose from a collection of tools, utilities or links.</p> <p>Example: Toolbars, Menus, Navigation links in web pages.</p>	
<p>2. Linear View PU</p> <p>Description: User can view or edit linear data as text, graphic or tables.</p> <p>Example: Text docs, graphic, web pages, grids, trees.</p>	
<p>3. Form PU</p> <p>Description: User can fill in form of any type. Example: Registration forms, login forms, property sheets.</p>	

Table 5.1: Examples of Presentation Units

5.6 Prototype Elements

Prototypes are made up of several elements that represent the actual UI components. So far, these prototype elements were designed directly on paper and were not standardized. Standardizing the prototyping elements prevents many useless discussions on detailed design issues (Lauesen, 1997). In this paper, we propose several prototype elements that cover most of the user-tasks.

5.6.1 Development for PE Symbols

The symbols for Prototyping Elements (PEs) are made with the idea in mind that prototypes should look as neat as possible, but yet, simple to make. The symbols cover most of the tasks that are to be performed on a UI. However, they are not the same as real user-interface elements. The real user-interface elements tend to give look-and-feel of tangible objects and cover every aspect of user-interaction. Each of these prototype elements corresponds to a particular PU symbol (section 5.5). These symbols could be drawn with hand on a paper prototype, or in case of software tool could be placed (as an image) on screen and re-sized. The position should correspond to that of PU of PUCM the prototype is derived from. The unique ID (in the form of PEx.y, where x is category number and y is an element in that category) identifies the PE in the Mapping Matrix we propose in next section.

For creating set of PEs, we studied user-interfaces of common desktop and web applications. We also received a document from Daimler-Chrysler that they shared with us for studying vehicle user-interfaces. By discussing these user-interfaces, we developed hierarchy of widgets (figure 5.3) based on interactions tasks that could be performed on widgets:

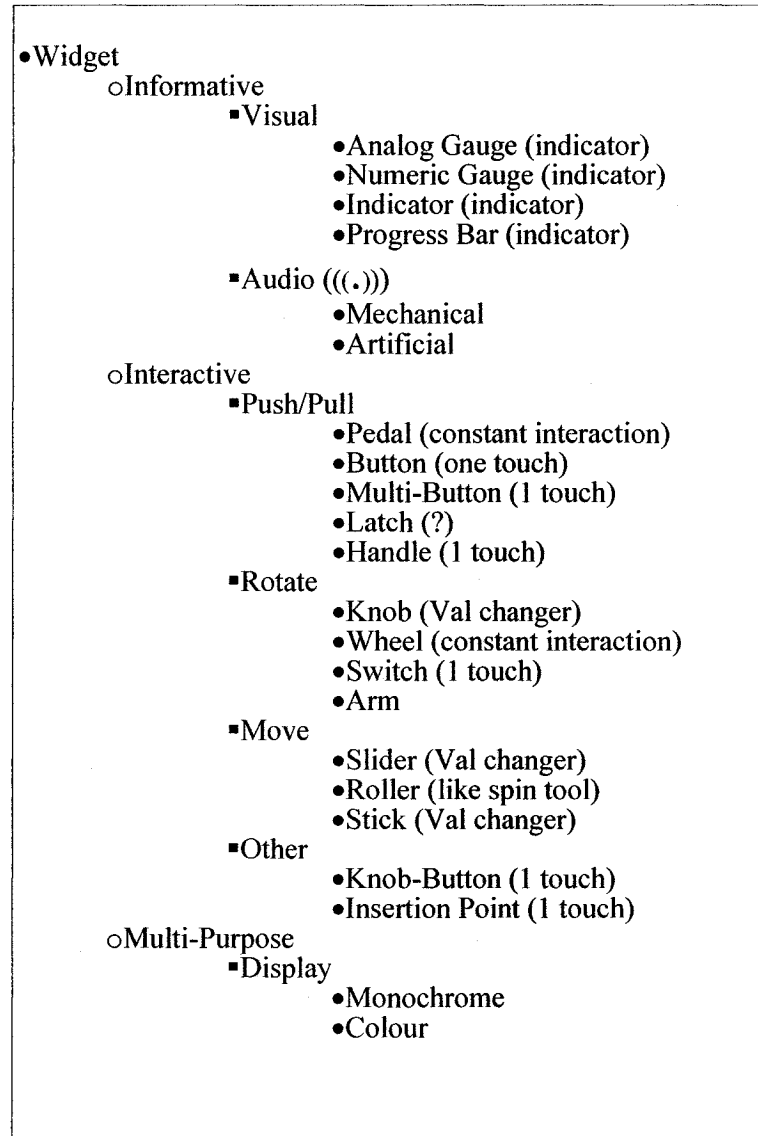
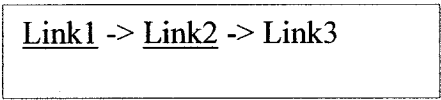


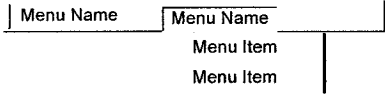

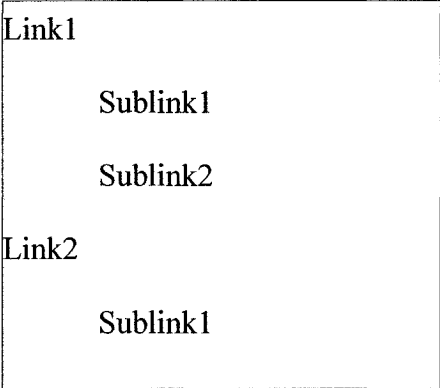
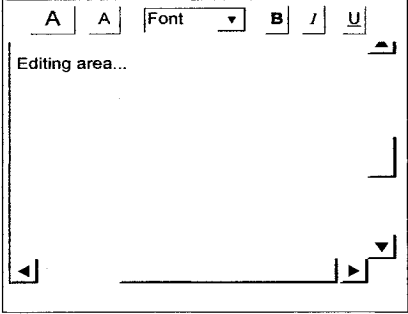
Figure 5.3: Initial car-dashboard widget hierarchy

When studying other user-interfaces (software applications), we found that some of this hierarchy applies to those widgets as well (e.g. Interactive -> Move -> Slider which is true for slider widgets in software applications as well). Furthermore, this hierarchy was also applicable on the PUs we created for PUCM (e.g. Audio Notification PU maps to Audio-> Mechanical and Audio -> Artificial in the hierarchy). Hence we created symbols of widgets for prototype (Prototype Elements in appendix C) according to the hierarchy and associated them with PUs for mapping in the tool (Mapping Matrix).

5.6.2 Some PE Symbols for Prototyping

Table 5.2 presents some examples of PEs. The complete set of our proposed PEs is presented in appendix C.

1. Tools Collection PE	
PE1.1 Navigational Path (series of hyperlinks) Description: Typically used in web-applications to show the navigational path followed. Each word is link and could be clicked to go to corresponding page.	

<p>PE1.2 Navigational hyperlinks</p> <p>Description: Typically used in web-applications to show the structure of navigation. Each word is link and could be clicked to go to corresponding page.</p>	
<p>PE1.3 Menu items in Menu bar.</p> <p>Description: Different features (or tools) are grouped and each group is accessible under its common name.</p>	
<p>PE1.4 Toolbar for collection of any tools.</p> <p>Description: A set of common tools are represented with a picture and displayed for easy access. Selecting a picture (by clicking or pointing) enables that tool or feature.</p>	
<p>2. Linear View PE</p> <p>PE2.1 Enhanced text editor</p> <p>Description: Complex component that allows entering and editing, as well as changing the attributes of the text.</p>	

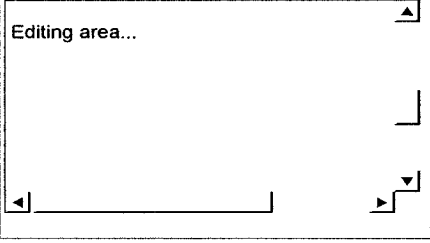
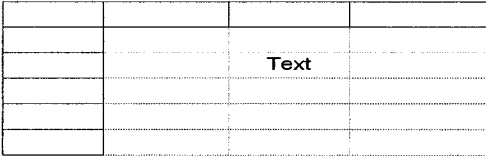
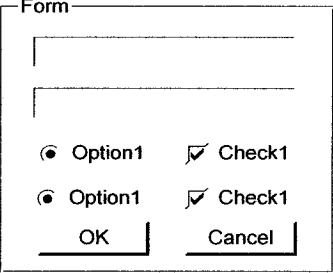
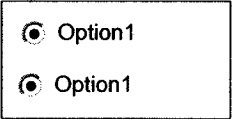
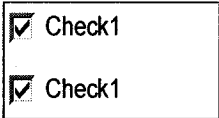
<p>PE2.2 Simple text-editor</p> <p>Description: Enables entering and editing text. Changing attributes of text is not possible.</p>	
<p>PE2.3 Grid view and editor</p> <p>Description: Enables entering and editing text in a grid/table.</p>	
<p>3. Form PE</p>	
<p>PE3.1 Form with different types of fields</p> <p>Description: Enables entering and editing values in a form containing multiple types of fields. There are buttons to submit the information and to clear the form.</p>	
<p>PE3.2 Form to select from mutually exclusive options.</p> <p>Description: Enables changing value of an attribute. User can choose one option.</p>	
<p>PE3.3 Form to select multiple options.</p> <p>Description: User can choose multiple options.</p>	

Table 5.2: Examples of Prototype Elements

5.7 Mapping Matrix - A tool for Deriving Prototypes using UCMs

Given the sets of Presentation Units and Prototype Elements, we will now present a tool in the form of a simple matrix that will help decide which PE should go for a given PU. We introduce a layer between these two notations namely “Intended action”. Since there could be more than one PEs for a PU (due to PU’s abstract nature), we have observed that an action is always *intended* to be performed on PU. This action is usually obvious from “responsibility” name in PUCMs. Depending on intended actions, we will help decide a suitable PE.

5.7.1 The proposed PU-to-PE Mapping Matrix

The proposed tool for prototype derivation is presented in table 5.3. For the sake of simplicity, we will refer to PUs by names whereas PEs by reference # (PE $x.y$ form, where x is category number and y is an element in that category). The extended set of PUs with pictures is presented in appendix B. The complete set of PEs is presented in appendix C.

Presentation Units (PU)	Intended Actions	Prototype Elements(PE)
1) Tools Collection PU	<ul style="list-style-type: none"> i. Hyperlink to other pages ii. Select from a set of tools 	<ul style="list-style-type: none"> PE1.1, PE1.2 PE1.3, PE1.4
2) Linear View PU	<ul style="list-style-type: none"> i. Edit text area ii. Edit grid iii. Edit graphics iv. view mixed (web) 	<ul style="list-style-type: none"> PE2.1, PE2.2 PE2.3 PE2.4 PE2.5
3) Form PU	<ul style="list-style-type: none"> i. Fill form (different types of fields) ii. Select one from many options iii. Select multiple from options 	<ul style="list-style-type: none"> PE3.1 PE3.2 PE3.3
4) System Dialog PU	<ul style="list-style-type: none"> i. Answer system's question ii. Take decision iii. Acknowledge information by system 	<ul style="list-style-type: none"> PE4.1 PE4.2 PE4.3
5) Indicator PU	<ul style="list-style-type: none"> i. View internal system info (desktop app) ii. View analog info 	<ul style="list-style-type: none"> PE5.1 PE5.2, PE5.3, PE5.4

	<ul style="list-style-type: none"> iii. View numeric info iv. View Boolean info v. View progress. 	<ul style="list-style-type: none"> PE5.5 PE5.6 PE5.7, PE5.8
6) Value Changer PU	<ul style="list-style-type: none"> i. Change numeric values ii. from selection iii. short-range continuous iv. long-range continuous v. short-range discrete vi. long-range discrete v. back n forth 	<ul style="list-style-type: none"> PE6.1, PE6.2 PE6.3 PE6.4 PE6.5, PE6.6 PE6.7, PE6.8, PE6.9 PE6.10 PE6.11
7) Constant Interaction PU	<ul style="list-style-type: none"> i. Keep/change direction ii. Keep/change speed 	<ul style="list-style-type: none"> PE7.1, PE7.2, PE7.3 PE7.4, PE7.5
8) One-touch Interaction PU	<ul style="list-style-type: none"> i. Toggle ii. Turn / move up or down iii. Insert, get-back iv. Hyperlink 	<ul style="list-style-type: none"> PE8.1, PE8.2 PE8.3, PE8.4, PE8.5, PE8.6, PE8.7, PE8.8, PE8.9 PE8.10 PE8.11
9) Audio notification PU	<ul style="list-style-type: none"> i. Input voice command ii. System alerts iii. System warns iv. Danger v. Feedback 	<ul style="list-style-type: none"> PE9.1 PE9.2 PE9.3 PE9.4 PE9.5, PE9.6

10) Navigable View PU	i. View textual artifact ii. View geographical artifact	PE10.1 PE10.2
11) Live View PU	i. View back ii. View sides iii. View anywhere	PE11.1 PE11.2 PE11.3
12) Multimedia PU	i. Watch and listen ii. Watch, listen and interact	PE12.1 PE12.2

Table 5.3: The Mapping Matrix

5.8 Concluding Remarks about the Mapping Matrix

The Mapping Matrix (table 5.3) is a tool to suggest Prototype Elements given the PUCMs. The PEs referred to in the matrix are given with detail in appendix C. These PEs cover most of the elements found in usual user-interfaces. However, the matrix could be expanded by adding more PEs for different new interfaces. Here, for example, the PEs related to common GUI and vehicles are given. The symbols were designed using simple graphical software, but for more sophisticated use, we suggest that these symbols be drawn by skilled artist in order to reveal necessary details of the gadget.

If new kinds of user-interfaces are to be supported, adding new PEs would be sufficient. The reason is that we have tried to generalize the Presentation Units (PUs of PUCMs) so as to cover most of the interaction possible by human with machines. Carefully designed PEs connected with the PUs through ‘intended action’ would be added to their respective columns in the table which could thus be used to suggest new PEs with a different interaction style suitable to the new kind of user-interface.

Another possible innovation would be to implement a software tool that would suggest PEs given the PUCMs. The matrix would then be implemented as the knowledge base of the software so that it can apply the matrix on PUCMs.

Chapter 6

Illustration of the Prototype Derivation

Process

6.1 Introduction

We have learned in previous chapters how a scenario that is made by observing users working in their environment can be represented in terms of Use-Case Maps (or UCMs in short). These UCMs if made following the SUCRE framework (Alsumait, 2004) are based on conceptual and physical models. From the physical UCMs, or PUCMs, we can make prototypes which are useful tools to illustrate the UI that is not developed yet and to verify the requirements as understood by developers. The process of derivation of prototype from scenario is presented in previous chapter. In this chapter, we illustrate how the process of derivation of prototypes from scenario helps in deriving prototype.

We present some scenarios that developers could create after a field study and then take one as a sample scenario to illustrate the process. The scenario is borrowed from vehicle industry and is very practical in today's modern systems in vehicles. The scenario is written in plain English language as most of the scenarios are first written. This approach is a characteristic of User-Centered Design and it is much used practice of contextual inquiry.

6.2 An introduction to dashboards and GPS

Dashboards are essentially the *control panels* of vehicles (figure 6.1). Drivers have to interact with the widgets on the dashboard to control the vehicle. Dashboards are focus of research in car industry and many new interactive systems are being explored to enhance user-interface of vehicle dashboards (Marcus et al., 2003).

Global Positioning Systems (GPS) are satellite-based systems for tracking position of objects on earth. They are winning popularity in car industry for way-finding and related services (circled part in figure 6.1). The main purpose of these GPS will be to guide the user on the roads, given the destination location. However, there is no limit on its use once we start thinking out-of-the-box, for example, finding a utility on a highway etc.



Figure 6.1: GPS in a car dashboard (courtesy

<http://tiger.towson.edu/users/nsharm1/>)

6.2.1 Scenarios related to dashboards and GPS

There could be several scenarios related to dashboard and GPS in vehicles. Here we state three such scenarios. For illustrating the complete process of prototype derivation, we will take another scenario (section 6.3.2).

Scenario 1: Mrs. Z was driving her children to mountain for a ski-trip one day in winter. Since weather was cold, children insisted on having a hot-chocolate and she too was feeling tired and wanted to have a cup of coffee. They did not know where to buy these because they were in the middle of highway. Mrs. Z decided to use the GPS in her car. She turned on the GPS and chose to search for utilities on highway. She found a coffee shop after 5 minutes drive. She noted that which exit she will need to take and turned the GPS off and drove till the coffee shop.

Scenario 2: Dr. Q was stuck in a maze of streets in a suburb he was visiting to see his aunt. He did not know which street to take to get to the highway and the street signs were broken due to last week's storm. He decided to use his GPS to know, first of all, where he is exactly and then which streets to take to get to the highway.

Scenario 3: Sergeant D was patrolling in his police car and was feeling terribly bored. He turns the radio on, seeks his favorite channel, adjusts volume and starts enjoying the football match commentary.

6.3 Step 1 – Scenario Elicitation

We will illustrate the process of scenario creation in terms of figure 3.5. The first step in scenario creation is Elicitation. We do this in several steps as detailed below.

6.3.1 Background of the Scenario

The scenario we take for illustration is related to GPS like we discussed in section 6.2. The character used in this scenario, *Sameer*, is a young *montrealer* living in downtown. One of his friends lives in *Longueuil*. He wants to visit his friend but has no idea how to get there. Luckily he has GPS computer in his car and he decides to use it. The scenario starts when *Sameer* sits in the car and ends when he reaches his friend. There could be other associated scenarios related with this one, but for the sake of simplicity we consider the simplest case.

6.3.2 The Scenario

On starting his car, Sameer turns on the GPS. He chooses to enter the street address of his friend that he asked on phone last night. He wants to give the system the full address but can't read the street number correctly, so he decides to give the postal-code only (which is unique throughout Canada). Sameer tells postal code to the system. GPS shows a map with road directions. He views the map in detail and sets the GPS to be always available; that is, the system tells him the road directions while he drives the car. Sameer reaches his friend's house safely and in time.

6.3.3 Capturing Tasks/Actions from Scenario

A scenario can be broken down for capturing tasks and actions that need to be performed. This practice will help in making better UCMs. We will analyze this scenario on the basis of task that the user is performing, where it starts and where it ends and the responsibilities (actions) user must perform to complete the task.

The ‘responsibilities’ are atomic actions in the UCM terminology. These actions are taken from the scenario itself. Some actions are obvious from the scenario while the others are implied and not so obvious.

Task:

To find way to reach friend’s house.

Start of Scenario:

Starting the car

End of Scenario:

Reaching friend’s house

Responsibilities:

Responsibility	Extracted from Scenario's part...
1. start GPS	Sameer turns on the GPS screen...
2. choose to add address	chooses to enter the street address of his friend...
3. Enter address	either enter the full address or only the postal-code...
3.1: enter full address	
3.2: enter postal-code	
4: view map	views the map in detail...
5: set 'guide-mode'	sets the GPS to 'guide-mode'...
6: follow map	map remains on his screen while he drives*

Table 6.1: Analysis of Scenario

* Note the responsibility # 6 which is not written verbatim in the scenario but is understood by reading the scenario carefully.

6.4 Step 2 – Scenario Specification

The second step in making scenarios according to figure 3.5 is Scenario specification. We will specify the scenarios in terms of UCMs as described in SUCRE framework in section 4.3.

6.4.1 Use-Case Maps

For the scenario given above we will make Use-Case Maps (Conceptual and Physical) to specify the scenario in more manageable and accurate diagrams.

The Conceptual Use-Case Map will show the responsibilities that are to be performed on different components to accomplish the task. The line shows the path taken and the crosses over the line shows the responsibilities over the components.

The Physical Use-Case Map will show how these responsibilities are performed on a system presentation level, rather than abstract (or conceptual) components as seen in CUCMs.

1) Conceptual Use-Case Map or CUCM

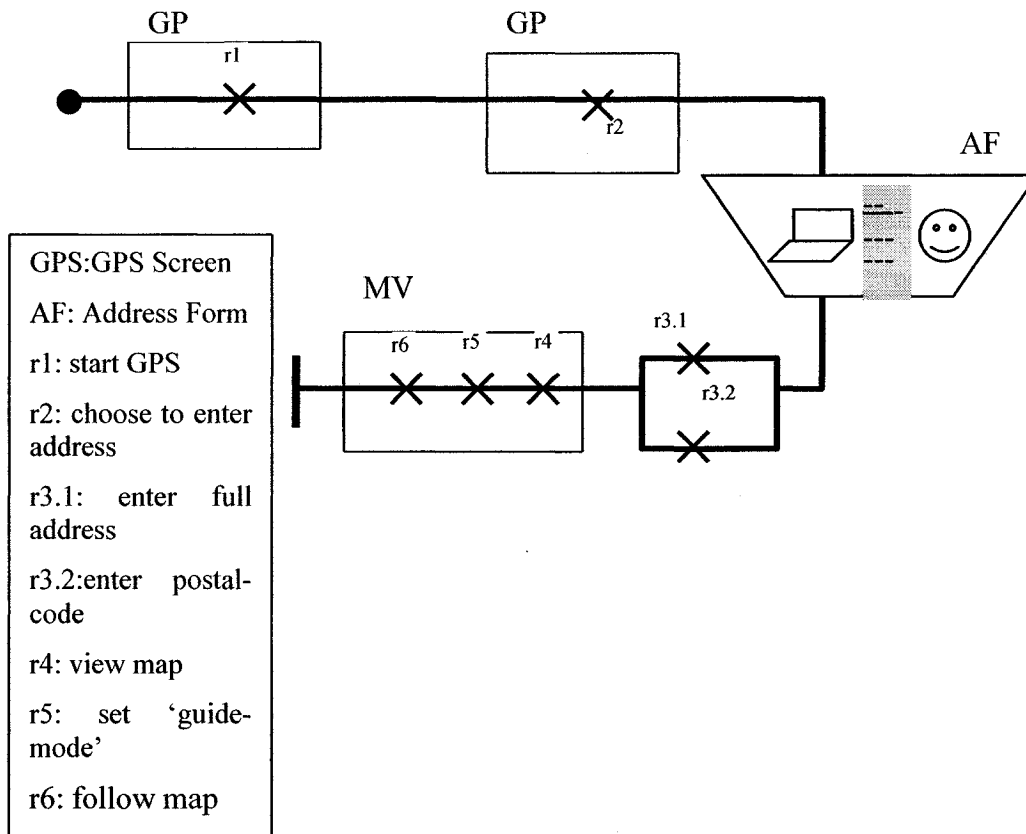


Figure 6.2: Conceptual Use-Case Map for Scenario

The Conceptual Use-Case Map (CUCM) in figure 6.2 shows how the task is accomplished by performing different responsibilities. Each responsibility is marked with a cross on the line in an abstract component which represents some part of the interface.

The filled circle at one end of the line shows the beginning of the scenario. As the line progresses, the responsibilities are performed and the line ends with a vertical bar which tells of the end of the scenario. The first responsibility (r1: Start GPS) is done on the GPS

itself (GPS component) when the screen is blank. Second responsibility (r2: choose to enter address) is also performed on GPS which is just started. After second responsibility is performed, the user is shown an 'address form' (AF component). User has then a choice to perform one of two actions as third responsibility (r3.1 or r3.2). After doing that, further responsibilities (r4, r5, r6) are done on a 'map view' (MV component).

2) Physical Use-Case Map or PUCM

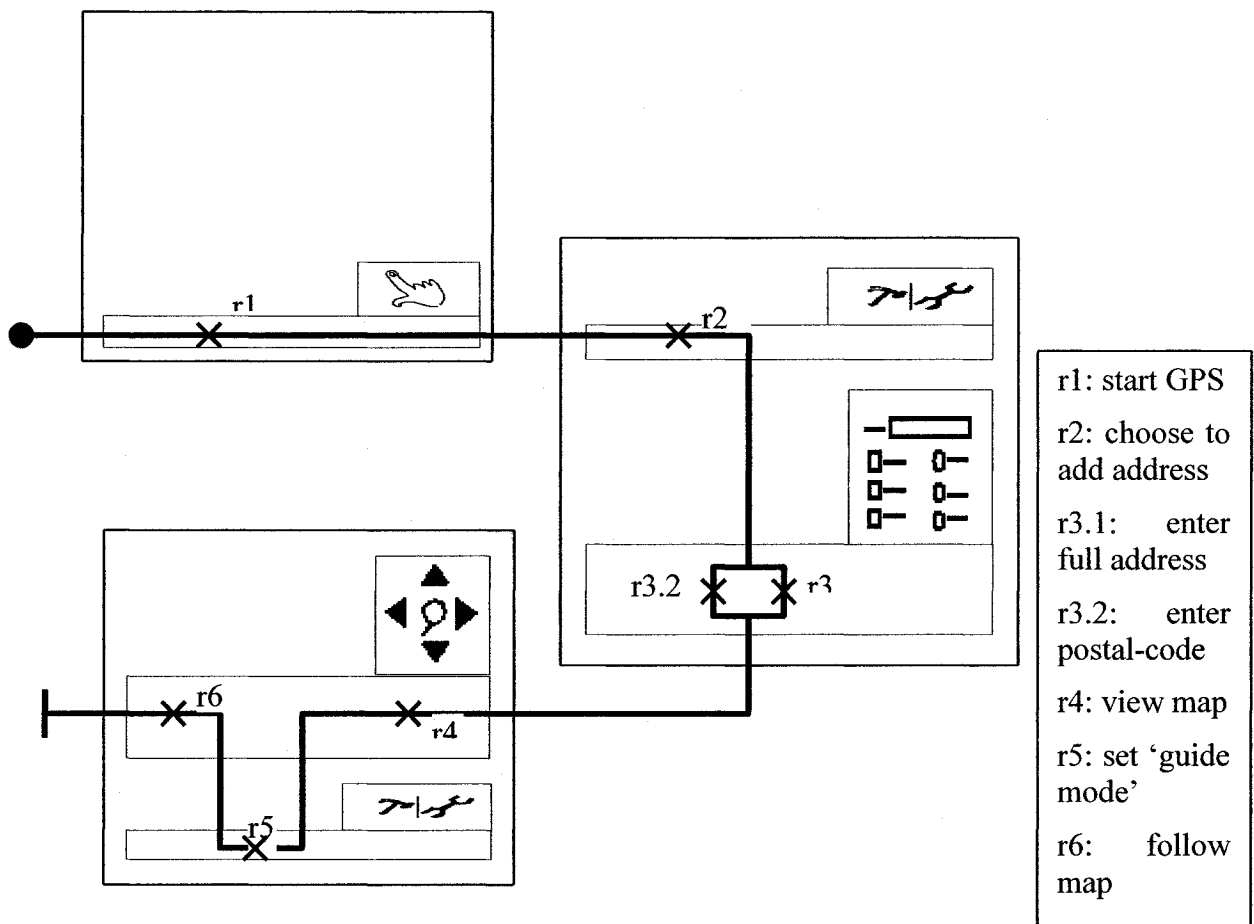


Figure 6.3: Physical Use-Case Map for Scenario

In the Physical Use-Cape Map (PUCM) in figure 6.3, the scenario is represented with lesser abstraction (closer to the user-interface layout). The flow of the scenario is still represented with thick line starting with a filled circle and ending in vertical bar.

The responsibilities are also performed in components but these components are now placed on a rectangle depicting the screen. Real-like widgets are still not present on this PUCM, rather, the 'presentation units' or PUs (rectangles with an icon on top-right corner) show parts of the screen. The PUs shown in figure 6.3 are drawn from the set of extended PUs proposed in this thesis presented in Appendix B.

The cross-marks depicting responsibilities now in PUs showing that associated action will be performed on that presentation unit of the UI. All the responsibilities remain the same as of CUCM of figure 6.2 and table 6.1.

6.5 Step 3 – Scenario Analysis

The third step in figure 3.5 is Scenario Analysis. As discussed in the figure, the analysis of the scenario could be performed with the help of a prototype. Therefore in this section, we make a storyboard prototype with the help of prototype derivation tool *Mapping Matrix* presented in table 5.1 in chapter 5. The mapping matrix, as discussed in section 5.7, takes input the names of Presentation Units that are found in PUCM and the *intended actions* from responsibilities (as in table 6.1) and suggests the Prototype Elements. This is done manually (by looking up the table for corresponding PUs in the table). The process of looking up of can be seen as in figure 6.4:

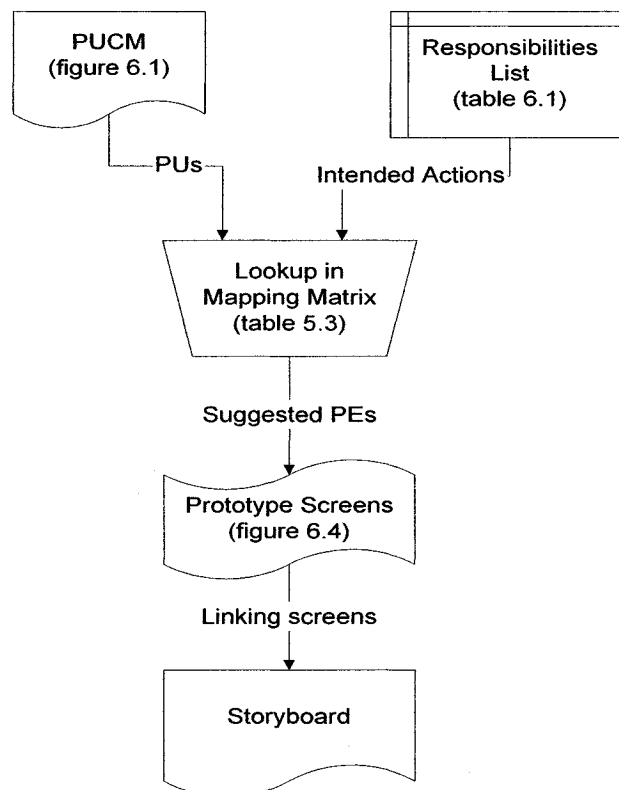


Figure 6.4: Application of Mapping Matrix

6.5.1 Applying Mapping Matrix

After making the Physical Use-Case Map, we can now refer to the Mapping Matrix of section 5.7 (as described in figure 6.4) and see which symbols we should put on Paper Prototype.

Presentation Unit (PU)	Intended Action	Suggested PE
One-touch interaction PU	turn on (toggle) the GPS screen	PE8.1 or PE8.2
Tools Collection PU	select from a set of tools	PE1.3 and/or PE1.4
Form PU	fill form	PE3.1
Navigable View PU	view geographical artifact	PE10.2

Table 6.2: Applying Mapping Matrix to PUCM

The result of applying Mapping Matrix is given in table 6.2. Based on PUs of PUCM in figure 6.3, we checked the intended actions to be performed on the PU. Taking the PU and the intended action over it, we mapped on Prototype elements. For each PU and its corresponding intended action, we found at least one PE (figure 6.5).

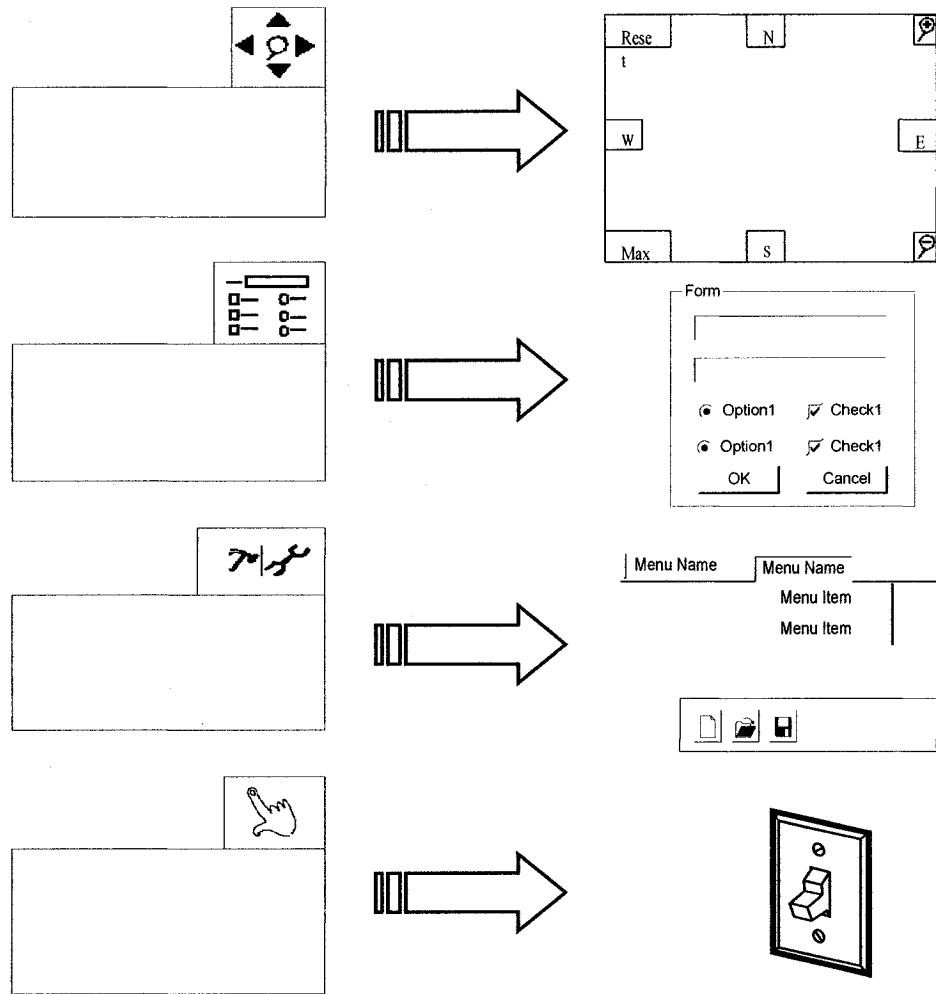
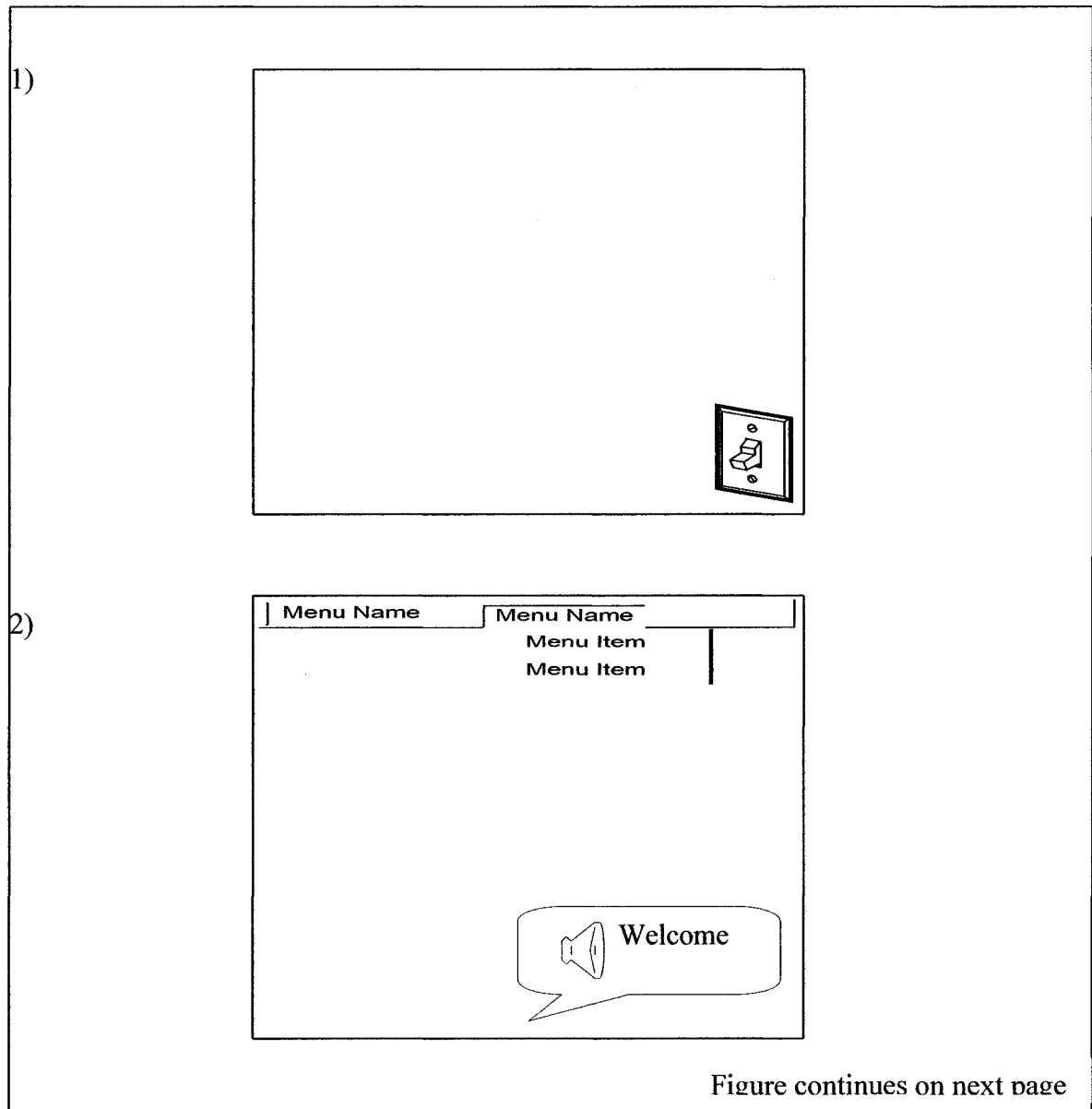


Figure 6.5: Suggested Prototype Elements (PEs) for Presentation Units (PUs)

Based on the result of applying Mapping Matrix (table 6.6), we will make a Paper Prototype using the suggested Prototype Elements (PEs). The position of the PEs on the screen is governed by the position of the corresponding PU in PUCM.

6.5.2 Storyboard Prototype

Following are the screens of suggested prototype (figure 6.6). These pictures are separate and are not connected with the cause-effect relationship.



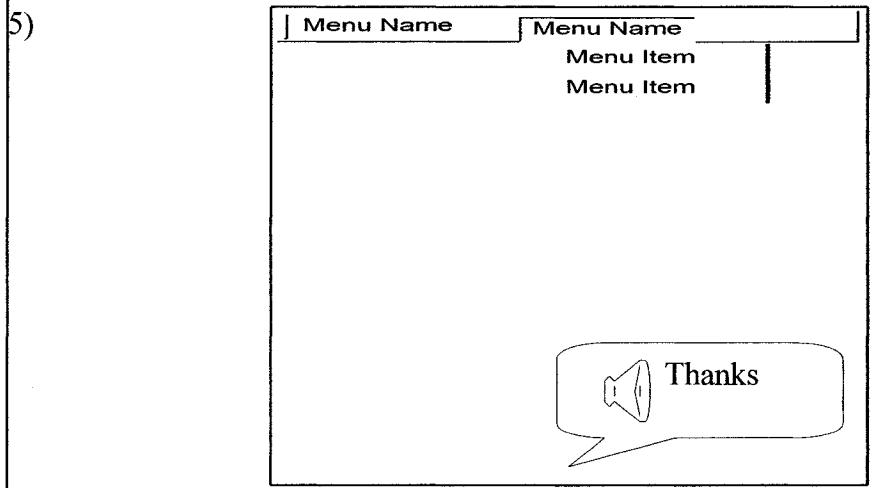
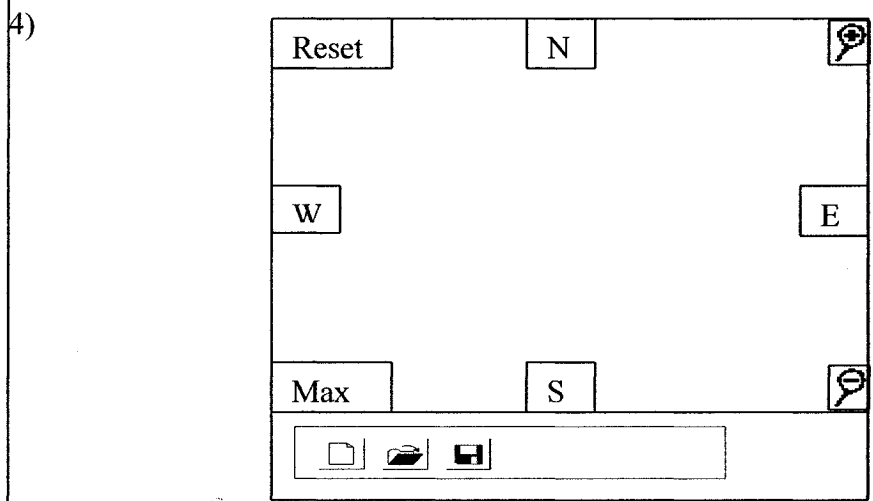
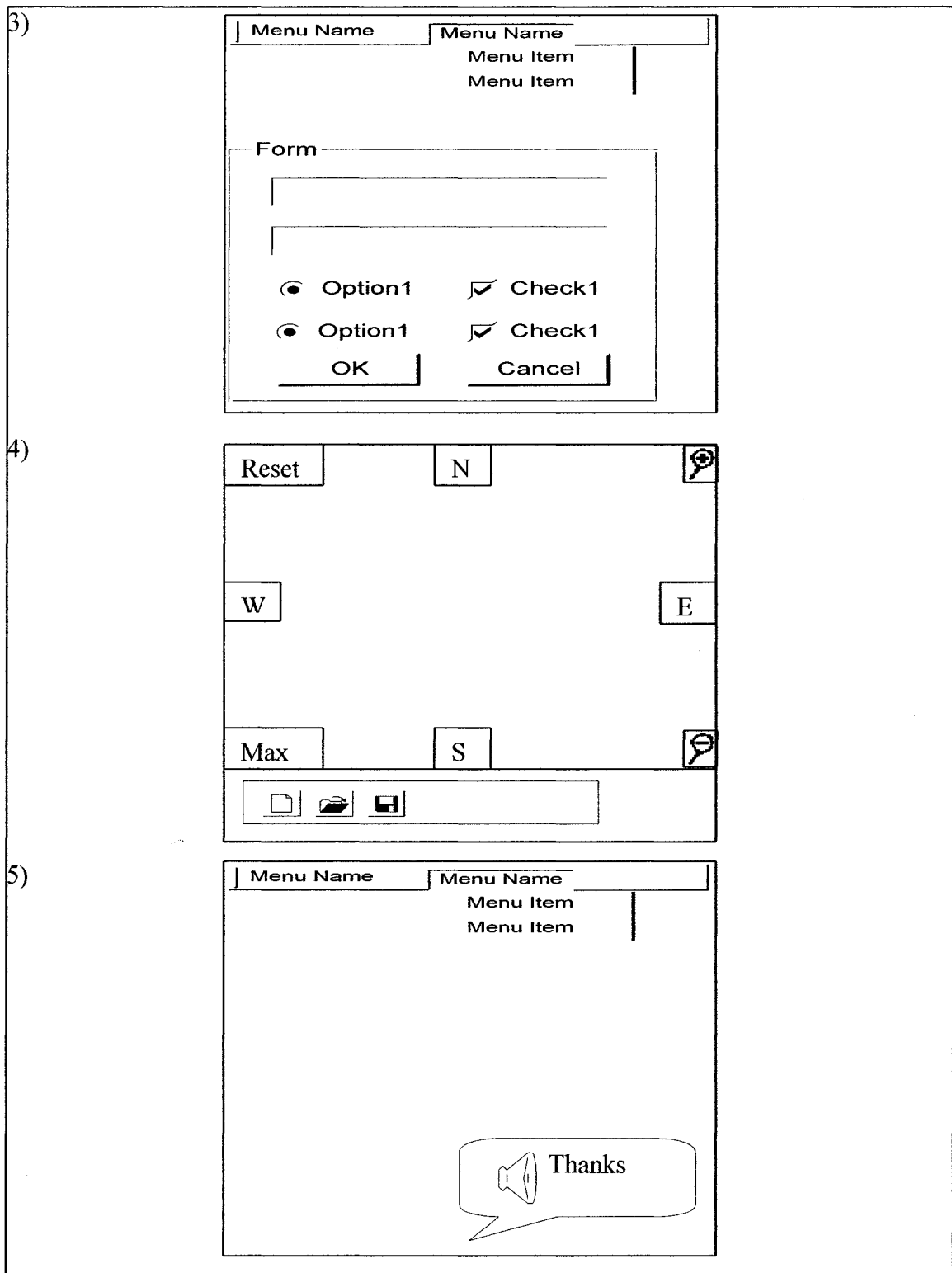


Figure 6.6: Screens of Prototype

These pictures when connected with each other with arrows, labeled with the actions that leads from first to next picture gives the storyboard as in figure 6.7 below.

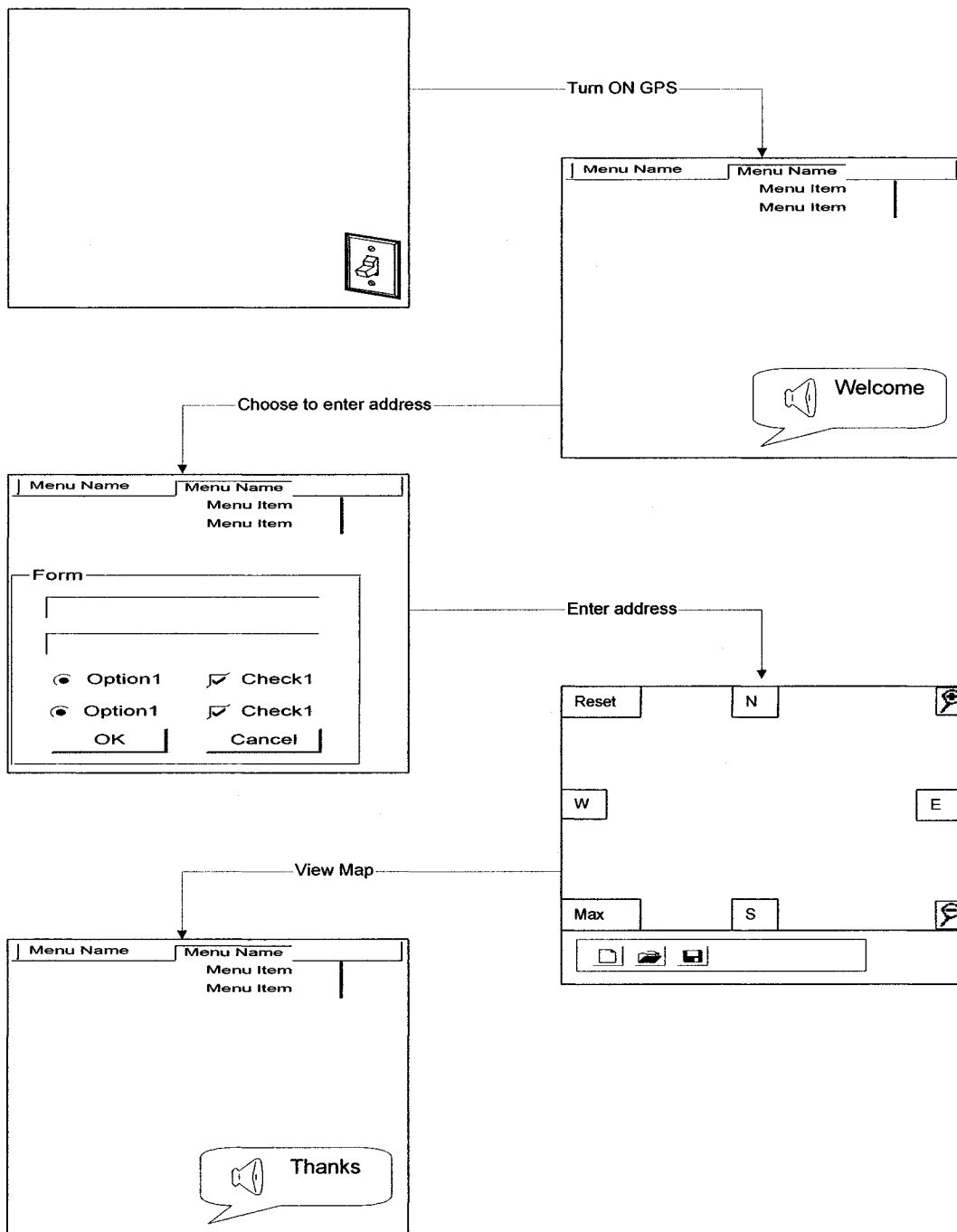


Figure 6.7 Final Storyboard

6.5.3 Explanation of the Storyboard

The storyboard in figure 6.7 could be explained with the following set of actions derived from the scenario. The actions at this stage are no more abstract and stated in terms of user-interface (or prototype) elements.

The storyboard starts with the picture of GPS in Off (or idle) state. When it is turned On using the switch, a welcome message is given to the user and user can chose from a menu an action to perform.

When user selects to enter an address, they are presented a form where they can enter the address. When the user presses OK button in the form, a map is shown.

User can navigate through the map and choose actions using the set of tools (a toolbar) below the map.

When the user finishes viewing the map (on arriving at the destination) the map goes from the screen and a 'Thank You' greeting is given to the user.

6.6 The storyboard and UI requirements in scenario

The storyboard fulfills the user-interface requirements in the scenario. As seen in table 6.1, the UI needs to have components (or widgets) that let the user:

- 1) Turn the GPS on
- 2) Choose from options on starting the GPS
- 3) Lets the user enter the address
- 4) Lets the user view the map
- 5) In the end remove the map from screen

All these requirements are fulfilled in the storyboard (figure 6.7). It also lets the user see the sequence of the screens to have an idea of the flow of scenarios.

6.7 Conclusion of Illustration

In this chapter, we illustrated the process of deriving prototype starting from a scenario. We first stated the scenario in plain English language and then captured major elements of the scenario. Then we represented the scenario in physical and conceptual UCMs as discussed in SUCRE framework. Later, we used the prototype derivation process proposed in chapter 5 to create a storyboard prototype.

Stating the scenarios and deriving prototypes from them is a major milestone in roadmap (figure 3.2) and is an important complementing part in SUCRE framework.

Chapter 7

Conclusions and Suggested Future Research

In this thesis, we proposed a roadmap for usability requirements engineering with includes artifacts and activities from agile methods. This roadmap is geared towards producing usability requirements using methods that are centered on users and their work context. Verification of gathered requirements is done using prototype derivation and evaluation. In this proposed roadmap, requirements are mainly represented in terms of scenarios that are represented as use case maps. The proposed roadmap use SUCRE, a framework proposed by the Human-Centered Software Engineering Group, for scenario specification and analysis.

Representation of scenarios using use-case maps has an added advantage. Using the conceptual and physical models of Use-Case Maps as presented in SUCRE framework, we proposed a process to derive prototypes from scenarios. This process helps designers create storyboard prototypes by suggesting standard user-interface objects. The derivation process uses a tool (in the form of a table) named Mapping Matrix which maps symbols (Presentation Units) in SUCRE framework's physical use-case maps to user-interface symbols on prototypes (Prototyping Elements). These two sets of symbols are created and named (Presentation Units of SUCRE are enhanced and extended) especially for this tool and they cover most of the interaction tasks.

The agilized roadmap for user-centered requirements and the process for prototype derivation through scenarios are interlinked. The scenario creation and analysis (which are major milestones in roadmap) through SUCRE framework is complemented with the prototypes which help evaluate the requirements. The roadmap and prototype derivation process are the major results of the research made for this dissertation.

7.1 Major Contributions

This thesis investigates a user requirements engineering approach that intimately combines user-centered design and agile philosophies. These philosophies are proposed by different people at different times, yet there are some points where we find synergy in these ideas. The major contributions of our research are outlined below.

- *A roadmap for User-centered requirements engineering* which incorporates *agile* aspects. This roadmap, detailed in chapter 3, describes the major milestones to gather and evaluate user requirements using prototypes and scenarios. With ideas from agile philosophy incorporated in the roadmap, it ensures the process of requirements engineering is fast, iterative, involves users at every major milestone and involves less documentation.

- *Categorization of interaction tasks* so that similar tasks could be represented with a symbol in physical use-case maps. This categorization could be used in the development of style-guides for widgets. We used this categorization for development of collection of Presentation Units and Prototype Elements presented in appendices B and C respectively.

- *A process to derive prototypes from scenarios*. The process of derivation takes scenarios as starting point and through use-case maps and Mapping Matrix tool, generates a

storyboard prototype. This process supplements SUCRE framework which suggests prototyping after detailed description of use-case maps modeling.

These contributions, we believe, will help the requirements engineering community to effectively gather requirements from users and their environment and verify those requirements using prototypes. These contributions also encourage future research as discussed in next section.

7.2 Suggested Future Research

Our research could be followed up and advanced to achieve more innovative results. We suggest the following agenda for improvements, refinement, and validation:

- *Validation of the Roadmap.* The roadmap presented in this thesis (figure 3.2) should be validated in an industrial environment. The steps of roadmap should be followed by analysis team and potential users of the product should be involved from the beginning. The helping forms and templates provided in appendix A would be beneficial especially in vehicle industry. In our research group at Concordia University, the project sponsored by Daimler-Chrysler (Vehicle Company) could prove to be the most suitable test benchmark for the roadmap. The forms and templates of appendix A are tailored for requirements engineering in a vehicle company like Daimler-Chrysler. Applying the roadmap could surface some suggestions to improve the roadmap.

- *Software Tool Support for Prototype Derivation.* The process for prototype derivation from scenarios should be supported by a software tool. This tool should implement the Mapping Matrix (table 5.1) as its knowledge base and let the users draw the Physical and Conceptual UCMs. This tool would have the Presentation Units (PUs) and Prototype Elements (PEs), from appendices B and C respectively, integrated and suggest users PEs based on the Mapping Matrix.

- *Extend PEs collection for different kinds of interfaces.* The Mapping Matrix maps one or many PEs for a given PU. Some PEs are typically used in only one type of interface (e.g. desktop or web applications, vehicle dashboards, etc). If the system under development requires different (or even multi-) interfaces, the prototypes must reflect elements of that interface. To achieve this, the PEs collection will have to be extended with new elements. Interaction tasks should be analyzed for those interfaces and new PEs should be drawn.

References

- Abrahamson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002) Agile software development methods - Review and analysis, ESPOO 2002, VTT Publications, pp.107.
- Achour, C. B. (1998) *Writing and correcting textual scenarios for system design* In Proceedings of the Ninth International Workshop on Database and Expert Systems Applications, 26-28 Aug. 1998, Vienna, Austria, pp. 166 - 170.
- Alsumait, A. (2004) *User Interface Requirements Engineering: A Scenario-Based Framework*, PhD. Thesis in Dept. of Computer Science and Software Engineering, Concordia University, Montreal.
- Alsumait, A., Seffah, A. and Radhakrishnan, T. (2003) *Use Case Maps: A Visual Notation for Scenario-Based User Requirements* In Proceedings of the 10th International Conference on Human - Computer Interaction, June 22 - 27, Crete, Greece.
- Amyot, D. (1998), *Use Case Maps web page*, accessed: May 2004, available at: <http://www.usecasemaps.org>, September 30, 2001
- Amyot, D. (2000) *Use Case Maps as a Feature Description Notation* In Proceedings of the FIREworks Feature Constructs Workshop, May 2000, Glasgow, Scotland, UK, pp. 18.
- Anderson, J., Fleek, F., Garrity, K. and Drake, F. (2001) 'Integrating usability techniques into software development', *IEEE Software*, vol. 18, issue. 1, pp. 46-53.

- Bai, X., Tsai, W. T., Paul, R., Feng, K. and Yu, L. (2002) *Scenario-based modeling and its applications* In Proceedings of the Proceedings of the Seventh International Workshop on Object-Oriented Real-Time Dependable Systems, 7-9 Jan. 2002, pp. 253 - 260.
- Beck, K., Anderson, A., Beattie, R., Bryant, D., DeArment, M., Fowler, M., Fronczak, M., Garzaniti, R., Gore, D., Hacker, B., Hendrickson, C., Jeffries, R., Joppie, D., Kim, D., Kowalsky, P., Mueller, D., Murasky, T., Nutter, R., Pantea, A. and Thomas, D. (1998) 'Chrysler goes to "Extremes"', *Distributed Computing*, issue. Oct. 1998, pp. 24-28.
- Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001), *Agile Manifesto*, accessed: March 2004, available at: <http://www.agilemanifesto.org/>
- Beyer, H. and Holtzblatt, K. (1998) *Contextual Design*, Morgan Kaufmann Publishers, Inc., San Francisco, CA.
- Buhr, R. J. A. (1998) 'Use Case Maps as Architectural Entities for Complex Systems', *IEEE Transactions on Software Engineering*, vol. 24, issue. 12, pp. 1131-1155.
- Carrol, J. M. (1999) *Five reasons for scenario-based design* In Proceedings of the 32nd Annual Hawaii International Conference System Sciences, 1999. HICSS-32, 5-8 Jan. 1999, pp. 11.
- Carroll, J. M. (1999) *Five reasons for scenario-based design* In Proceedings of the 32nd Annual Hawaii International Conference System Sciences 1999 - HICSS-32, 5-8 Jan. 1999, Maui, Hawaii, vol. 3, pp. 11.

- Carroll, J. M. (2000) *Making Use: Scenario-based design of Human-Computer Interactions* In Proceedings of the Designing interactive systems: processes, practices, methods, and techniques, December 2000, New York, NY, USA, ACM Press New York, USA, pp. 4.
- Carroll, J. M., Rosson, M. B., Chin, G. J. and Koenemann, J. (1998) 'Requirements development in scenario-based design', *IEEE Transactions on Software Engineering*, vol. 24, issue. 12, pp. 1156 - 1170.
- Cockburn, A. (2000) *Characterizing People as Non-Linear, First-Order Components in Software Development* In Proceedings of the 4th International Multi-Conference on Systems, Cybernetics and Informatics, June 2000, Orlando, Florida, pp. 19.
- Constantine, L. L. and Lockwood, L., A.D. (2002) 'Usage-Centered Engineering for Web Applications', *IEEE Software*, vol. 19, issue. 2, pp. 42 - 50.
- Cox, K. (2000) *Fitting scenarios to the requirements process* In Proceedings of the 11th International Workshop on Database and Expert Systems Applications 2000 (DEXA'00), 4-8 Sept. 2000, Greenwich, London, U.K., IEEE Computer Society Press, 2000, pp. 995 - 999.
- Ferre, X. (2003) *Integration of Usability Techniques into the Software Development Process* In Proceedings of the International Conference on Software Engineering - ICSE 2003, May 3-10, 2003, Portland, Oregon, USA, pp. 28-35.
- Fowler, M. (2000), *The New Methodology*, accessed: March 2004, available at: www.martinfowler.com/articles/newMethodology.html, April 2003
- Greenberg, S. (1998), *Prototyping for Design and Evaluation*, accessed: August 2004, available at: <http://www.cpsc.ucalgary.ca/~saul/681/1998/prototyping/survey.html>

How to develop Usability Goals (1996), accessed: 2004 available at:

http://www.stcsig.org/usability/resources/toolkit/use_gls.doc

John, B. E., Bass, L. and Adams, R. J. (2003) *Communication across the HCI/SE divide:*

ISO 13407 and the Rational Unified Process In Proceedings of the HCI

International, June 2003, Crete, Greece, pp. 5.

Jokela, T., Iivari, N., Matero, J. and Karukka, M. (2003) *The Standard of User Centered*

Design and the Standard definition of Usability: Analyzing ISO 13407 against

ISO 9241-11 In Proceedings of the Latin American conference on Human-

computer interaction, Rio de Janeiro, Brazil, ACM Press New York, NY, USA,

pp. 53 - 60.

Kutschera, P. and Schafer, S. (2002), *Applying Agile methods in rapidly changing*

environments, accessed: 2004, available at:

[http://jeckstein.com/papers/Agile%20Methods%20-](http://jeckstein.com/papers/Agile%20Methods%20-%20Steffen%20Schaefer%20&%20Peter%20Kutschera.pdf)

[%20Steffen%20Schaefer%20&%20Peter%20Kutschera.pdf](http://jeckstein.com/papers/Agile%20Methods%20-%20Steffen%20Schaefer%20&%20Peter%20Kutschera.pdf), July-23-2002

Landay, J. A., Lin, J., Newman, M. W. and Hong, J. I. (2000) *DENIM: finding a tighter*

fit between tools and practice for Web site design In Proceedings of the SIGCHI

conference on Human factors in computing systems, The Hague, The

Netherlands, ACM Press New York, NY, USA, pp. 510 - 517.

Landay, J. A. and Myers, B. A. (1995) *Interactive sketching for the early stages of user*

interface design In Proceedings of the SIGCHI conference on Human factors in

computing systems, Denver, Colorado, United States, ACM Press/Addison-

Wesley Publishing Co. New York, NY, USA, pp. 43 - 50.

- Lauesen, S. (1997) *Adding Usability to Software Engineering* In Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, INTERACT '97, 14th-18th July 1997, Sydney, Australia, Chapman & Hall 1997.
- Maguire, M. C. (1998), *RESPECT 5.3: User-Centred Requirements Handbook*, accessed: 31st March 2004, available at:
<http://www.ejeisa.com/nectar/respect/5.3/contents.htm>, 29 June 1998
- Marcus, A., Boehm-Davis, D. A., Green, P. A., Hada, H. and Wheatley, D. (2003) *The next revolution: vehicle user-interfaces and the global rider/driver experience* In Proceedings of the Conference on Human Factors in Computing Systems, Ft. Lauderdale, Florida, USA, ACM Press New York, NY, USA, pp. 708 - 709.
- Mayhew, D. J. (1999) *The Usability Engineering Lifecycle: A practitioner's handbook for User Interface Design*, Morgan Kaufmann Publishers, Inc., San Francisco, California.
- Merriam-Webster (1982), *Merriam-Webster Online Dictionary*, accessed: 2004, available at: www.m-w.com
- Miga, A., Amyot, D., Bordeleau, F., Cameron, D. and Woodside, M. (2001) *Deriving Message Sequence Charts from Use Case Maps Scenario Specifications* In Proceedings of the 6th Mitel Conference on Innovation in Applications & Technology (MICON2001), June 2001, Ottawa, Canada, August 2001, Springer-Verlag London, UK, pp. 268 - 287.
- Orr, K. (2004) 'Agile requirements: opportunity or oxymoron?' *IEEE Software*, vol. 21, issue. 3, pp. 71 - 73.

- Paetsch, F., Eberlein, D. A. and Maurer, D. F. (2003) *Requirements Engineering and Agile Software Development* In Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), 9-11 June 2003, IEEE Computer Society, pp. 308 - 313.
- Patton, J. (2002) *Hitting the Target: Adding Interaction Design to Agile Software Development* In Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications - OOPSLA 2002 Practitioners Reports, November 2002, Seattle, Washington, ACM Press New York, NY, USA, pp. 1 - ff.
- Pohl, K., Weidenhaupt, K., Jarke, M. and Haumer, P. (1998) 'Scenarios in system development: current practice', *IEEE Software*, vol. 15, issue. 2, pp. 34 - 45.
- Seffah, A., Alsumait, A., Radhakrishnan, T., Kotzé, P. and Poll, J. A. v. d. (2003) *Combining UCMs and formal methods for representing and checking the validity of scenarios as user requirements* In Proceedings of the Proceedings of the Annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology, September 2003, South Africa, South African Institute for Computer Scientists and Information Technologists Republic of South Africa, pp. 59 - 68.
- Seffah, A., Djouab, R. and Antunes, H. (2001) *Comparing and reconciling Usability-Centered and Use Case-Driven Requirements Engineering Process* In Proceedings of the 2nd Australasian conference on User interface, Queensland, Australia, IEEE Computer Society Washington, DC, USA, pp. 132 - 139.

- Sutcliffe, A. G., Maiden, N. A. M., Minocha, S. and Manuel, D. (1998) 'Supporting scenario-based requirements engineering', *IEEE Transactions on Software Engineering*, vol. 24, issue. 12, pp. 1072 - 1088.
- Sutcliffe, A. G. and Ryan, M. (1998) *Experience with SCRAM, a Scenario Requirements Analysis Method* In Proceedings of the Proceedings of the Third International Conference on Requirements Engineering, 1998., 6-10 April 1998, Colorado Springs, pp. 164 - 171.
- UPA (2002), *Usability Professionals Association website*, accessed: March 2004, available at: <http://www.usabilityprofessionals.org>
- Weber, M. and Weisbrod, J. (2003) 'Requirements Engineering in Automotive Development: Experiences and Challenges', *IEEE Software*, vol. 20, issue. 1, pp. 16 - 24.
- Wells, D. (2004), *Extreme Programming: A gentle introduction*, accessed: March 2004, available at: <http://www.extremeprogramming.org/>, February 28, 2004
- Z.Guan, J. and Luqi (2003) *A Software Prototyping Framework and Methods for Supporting Human's Software Development Activities* In Proceedings of the International Conference on Software Engineering - ICSE'03, May 3-11, 2003, Portland, Oregon, pp. 114-121.

Appendix A

Forms and Templates for the Roadmap

A.1 Vision Document

Tentative table of contents for Vision Document

1. Introduction

1.1 Purpose

1.2 Scope

2. Users of System

2.1 Highlights of user characteristics

2.2 Major tasks users accomplish

3. Environment of use

3.1 Context-of-use highlights

3.2 User-Support

4. Process

4.1 Process/Standards to follow

4.2 Client involvement

Each topic should typically take 2 to 4 lines of elaboration.

Any diagrams deemed useful should be included.

A.2 Forms for Field Study

A.2.1 Identifying Users and Stakeholders

While conducting field study, look for different ‘kinds’ of users. While you will encounter a variety of user groups, you might find some stakeholders also. Stakeholders are people who do not use the product directly but are involved as a middle-man between product developers and real users, e.g. Car-dealers, company manager, etc.

Form FS1.1 will help filtering different groups of users. (This form and other subsequent forms are designed for DC-team especially, so examples will assume users of vehicles.)

User ...	Yes	No
Is a Driver?	√	
Sits beside driver?		
Sits on backseat?		
A stakeholder? If Yes, Describe: _____		
Male?	√	
Female?		

Age <5 yrs?		
Age 5-18 yrs?	√	
Age 18-70 yrs?		
Age 70+ yrs?		
Experience of system? If Yes, Describe: _____		√
Exposure to related technology? If Yes, Level: ___ Driving trucks _____	√	
Has mental disability? If Yes, Describe: _____		√
Has physical disability? If Yes, Describe: _____		√
Any other constraints? If Yes, Describe: _____		

Form FS.1.1

For every person met during field study, fill up this table and on the basis of the data, make user groups. For example, Drivers aged 70+ yrs with no physical disability, User who sits on back seat and age less than 5 yrs, Drivers aged 18-70 yrs with physical disability, etc.

Once common characteristics are identified on the basis of data on FS1.1, name the groups and fill the following form (Maguire, 1998):

Group Name	Role in System or Use of System	Expand
Typical Driver	Will drive car, use most of the widgets himself most of the time. Have no apparent difficulty in operating widgets and understanding information.	√
Senior Driver	Will drive the car. Need proper concentration without distraction. Might have difficulty in using some widgets.	√
Child passenger	Will not drive car, will sit on back in special seat for added safety.	√
Car Dealer	Will help people buy new/used cars. Will have detailed information about features but will not use personally (otherwise shift to other user-group).	

Form FS1.2

Give a distinct name to each user group and explain in short their role in the system or how they will use the product. For those groups that user requirements will be described for, check under the expand column. For example,

Group: Senior Driver

Role: Will drive the car but not too often, might face problem in using some widgets...

Expand: checked

Group: car-dealer

Role: Will help users select cars of their needs, sell the car to users.

Expand: unchecked

A.2.2 User Characteristics

In this table, note down characteristics of different user-groups. Give an identification number to each characteristic so that it could be referred to later. Use one form for each group.

User group name: Senior Driver		
Characteristics	Potential requirements	Ref. #
Size of group:		
Age range:		
Gender distribution:		
Educational level:		
Physical limitations:		
Language & culture:		

<p>Experience with similar systems:</p> <p>Frequency of use:</p> <p>Likely concerns:</p>		
--	--	--

Form FS1.3

A.2.3 Social environment of use

A system is rarely used in seclusion from society. Social environment impacts a great deal on users using the system. Vehicles are driven by humans but also used by other passengers, so the overall social environment inside the vehicle and outside of it is important.

During field study, DC-team will observe how people and social behavior affect use of vehicles. To do this, following form will be helpful in collecting all relevant information.

To fill the form, first note down all the characteristics of social behavior then turn-by-turn, analyze what user requirements are important regarding that characteristic.

Characteristics	User Requirements	Ref. #
<p>Communication with outside world:</p> <p>Performance</p> <p>Monitoring:</p> <p>Performance Feedback:</p> <p>Group usage:</p> <p>Assistance availability:</p> <p>Interruptions from inside:</p> <p>Interruptions from outside:</p> <p>Stressful conditions:</p> <p>Safety (from accidents):</p>		

Security (from crime): Privacy: Entertainment: Emotional attachment: Duration of use: Traffic/road signs/signals:		
---	--	--

Form FS1.4

A.2.4 Users' Anecdotes

In this part, DC-team will record anecdotes that users tell about their driving experience. Anecdotes are short stories of some real events that happen to a person. Its very common among friends and family members when they are sharing their day to day interactions

with people and systems. These short stories will provide valuable information for gathering scenarios.

In the following form, write the short story told by users, record the people involved in the story (like driver's kids or policeman) and note the group users belong to:

Event Date: January 2003		(write approximated)
Event Time: Afternoon		(write approximated)
<p>Story:</p> <p>One day in last year winter, I was taking my grandfather for regular medical checkup to hospital. He was sitting beside me while I was parking my car just in front of another car. When I shifted to reverse gear and started reversing, I couldn't feel I was so close to the car on back due to snow on back screen; I hit that car and was embarrassed, although it not too hard a hit. I wish I had a device in my car that could tell me how far away I m from an object behind.</p>		
Actors involved:		User-group of actors:
Driver		Adult Driver
Grandfather		Senior Passenger

Form FS1.5

A.3 Usability Goals

Proposed Template for writing Usability Goals

Serial #:

Title

Description

Priority

Specific user group name (if any)

Measurement method (if any)

Usability factor(s) involved

Conditions (if any)

Benchmark tests(s).

Comments/Notes/Rationale.

Example:

Serial #: 2

Title: Door mirror adjustment

Description: User should be able to adjust the orientation of door mirror in 5 seconds without lowering the door glass.

Priority: Medium

User group: Driver

Measurement method: null

Usability factor(s) involved:

Conditions: Must not necessary to lower window.

Benchmark test: User will sit in driving position and use mirror adjuster (to be designed) to adjust the mirror as needed. Time will be noted and it should be no more than 5 seconds.

Notes/Rationale: In winter when a little wind entering through door glass is uncomfortable, user might need to adjust the door mirror which is an essential task for safe driving. The adjuster (a widget) should be in reach of the driver while driving and must be usable enough to let the driver adjust mirror without distraction.

A.4 Stating Scenarios

Scenarios can be stated formally in the following format:

Serial #:

Ref. of Usability Goal:

Other References:

Scenario Description:

Usability Aim:

Example:

Serial #: 4

Ref. of Usability Goal: UG-2

Other References: FS1.5.3 (User Anecdote)

Scenario Description:

On a hot summer day, children sitting on back-seats ask their father who was driving the car that the car is too hot and want him to turn on the car air conditioner. Driver winds up all windows and turns on the air conditioner. After a few minutes the car temperature became comfortable.

Usability Aim: 80% of users should wind up windows before turning on a/c. The widget to start a/c should be not more than an arm's distance for driver.

A.5 Task Analysis

Task definition and analysis involves identifying tasks and breaking them down into subtasks until they are decomposed to a set of simple actions. Following scheme could be used to do task analysis:

Identify tasks from the scenarios noted on previous step and list them down.

Sort and list main tasks that need to be performed and the Goal to achieve by performing these tasks (see Usability Goals step). Call this list of main tasks, Level-1

Focus on each task in Level-1 one by one and decompose this task into subtasks (i.e. things to do to perform task of level 1). Call this Level-2

Repeat step 3 on subtasks of Level 2 in place of Level-1 until you cannot further break down tasks. These atomic subtasks are called Actions.

Describe each actions and time consumed in performing these actions.

Example:

Following is a major task that is common while driving vehicles.

Level 1

Task 1: Taking a Left-turn

Goal: User must be able to turn on right/left side roads (90-degree turn) *safely* and quickly.

Level 2

Subtask 1.1: Change lane(s) towards leftmost lane.

Subtask 1.2: Turn left on the turning.

Level 3

Subtask 1.1.1: Turn-on the indicator of left side

Subtask 1.1.2: If vehicle is not too near, change lane(s) until reach leftmost lane.

Subtask 1.2.1: Turn-on the indicator of left side.

Subtask 1.2.2: Turn.

Level 4

1.1.1.1: Locate indicator widget (2 secs.)

1.1.1.2: Press/flip widget (<1 secs.)

1.1.2.1: Locate rear view mirror (or other widget) (<1 secs.)

1.1.2.2: See if another vehicle is near in left lane (3 secs.)

1.1.2.3: Make a decision whether to move (unknown time, depends on traffic)

1.1.2.4: If decide to move in 1.2.2.3 then turn steering wheel until car is in next lane.

Else, repeat from action 1.2.2.2

1.2.1.1: Locate indicator widget (2 secs.)

1.2.1.2: Press/flip widget (<1 secs.)

1.2.2.1: Stop at turning point to see if another vehicle/person is coming. (5 secs.)

1.2.2.2: If No, turn steering wheel until car is turn 90-degree around the turning point.

Else, wait until 1.2.2.1 is No. (unknown time, depends on traffic)

A.6 Study of Similar Systems

The aim of this step is to consider similar systems (may include competitive products) and to see what to include and, more importantly, what should not be included.

In the following form, we take each usability goal and see corresponding features in other systems. (Maguire)

Ref. #	Goal	Product Name/Company	Good Features	Bad Features
1	Doors must be closed before car starts moving	XYZ car	A light blinks with sound if any door is opened.	
2	Locking and unlocking doors should be easy and quick	ABC car		There is no central widget to lock/unlock doors near driver's seat.
...

Form SS1.1

A.7 Physical Environment

Following form helps in considering different aspects of environment relevant to vehicle usage and identifying user needs. This table should be filled for every relevant user-group separately. (Maguire, 1998)

Product name: Jeep		
User group: Drivers		
Ref.#	Physical Environment Characteristics	Usability Requirements
1	Typical location of use Roads and hilly, rocky, wet, sandy places (All Terrain Vehicle)	
2	Thermal conditions outside vehicle All weathers and atmospheres	Outside temperatures should not damage the vehicle and have a drastic affect on internal environment
3	Thermal conditions inside vehicle Milder than outside weather	Internal temperature should be comfortable enough.
4	Auditory condition Noise of natural phenomena (waterfalls, desert winds...) or noise of other vehicles on road.	Outside sound should not penetrate too much inside. Internal sounds should be audible even if it is noisy outside.

5	Light outside vehicle Natural light or darkness. Electric light if driving on road.	Headlights should be powerful enough to light up the path. Could be supplemented with extra lights.
6	Light inside vehicle Small lamps that consume less power	Internal lights should enable seeing all widgets clearly and things inside vehicle.
7	Vibration or instability Lots of vibration and instability due to varying terrains	User should feel enough comfortable be able to operate widgets or drive without difficulty.
8	Space and seating arrangement Limited space. Seats wide but not too comfortable	Seating should be comfortable and adjustable for drivers of different heights.
9	Health and Safety Danger of loosing balance, rolling over, being stuck in mud or sand.	Safety belts, air bags are necessary. There should be some communication device that can be used if needed in emergency.
10	Others

Form PN1.1

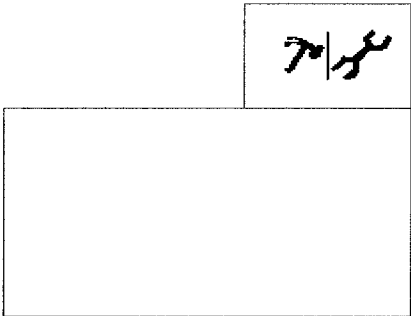
Appendix B



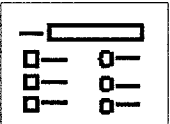
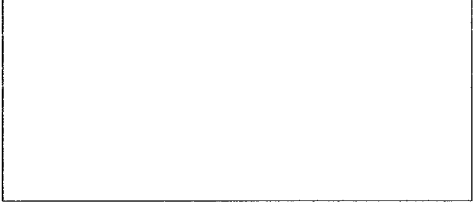
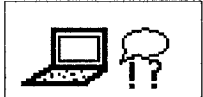
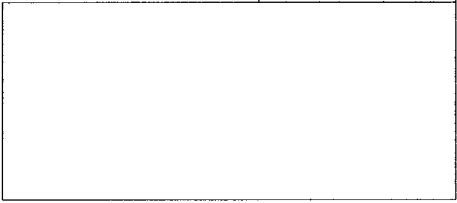
List of Suggested Presentation Units (PUs)

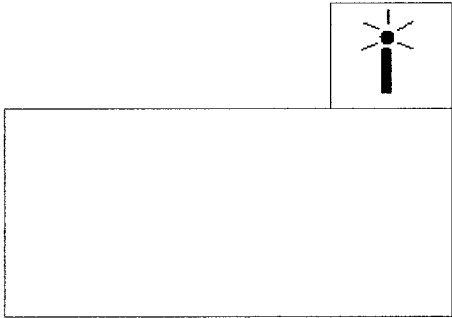
B.1 A Note about Presentation Units

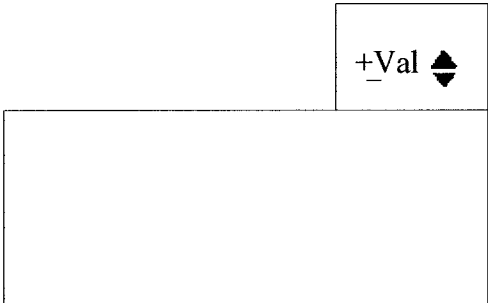
The Presentation Units were introduced in SUCRE framework. In this framework, these PUs were presented to explain the concept of PUCMs, therefore, those PUs did not cover all aspects of presentation. In this thesis, we attempt to cover all aspects of presentation and interaction, therefore the PUs presented here complement the original PUs presented in SUCRE framework

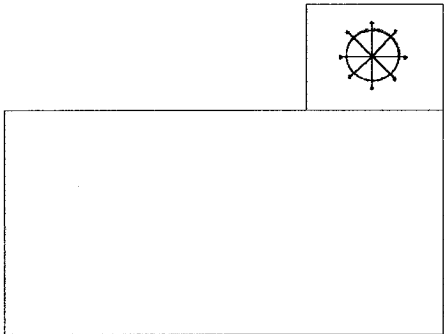
B.2 Presentation Units

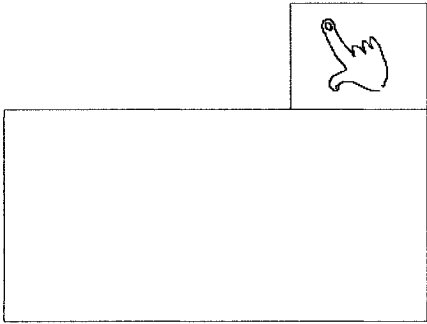
<p>1. Tools Collection PU</p> <p>Description: User can choose from a collection of tools, utilities or links.</p> <p>Example: Toolbars, Menus, Navigation links in web pages.</p>	 <p>The diagram illustrates a 'Tools Collection PU'. It consists of a large, empty rectangular box representing the collection area. Above the top-right corner of this box is a smaller, square icon containing a wrench and a screwdriver, symbolizing tools or utilities.</p>
--	--

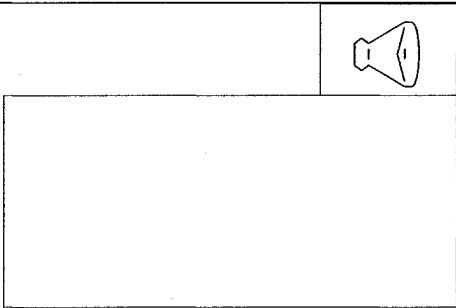
<p>2. Linear View PU</p> <p>Description: User can view or edit linear data as text, graphic or tables.</p> <p>Example: Text docs, graphic, web pages, grids, trees.</p>	 
<p>3. Form PU</p> <p>Description: User can fill in form of any type</p> <p>Example: Registration forms, login forms, property sheets.</p>	 
<p>4. System Dialog PU</p> <p>Description: User can respond to system's short messages.</p> <p>Example: Error message, task-completed message, restart-now-or-later message.</p>	 

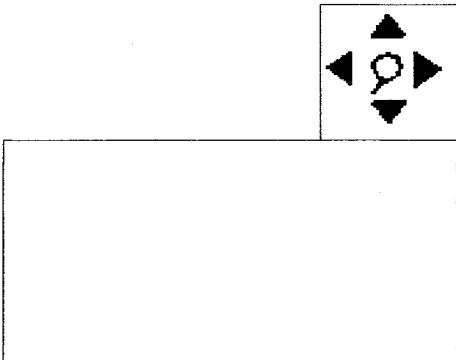
<p>5. Indicator PU</p> <p>Description: User can view little bits of information given by system.</p> <p>Example: system status, process progress, door open</p>	
--	--


<p>6. Value Changer PU</p> <p>Description: User can change values of different attributes of system.</p> <p>Example: Volume, speed, brightness.</p>	
--	---

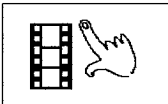
<p>7. Constant Interaction PU</p> <p>Description: User can interact with system constantly for longer periods of time.</p> <p>Example: Steering direction, controlling speed, game control.</p>	
--	--

<p>8. One-touch Interaction PU</p> <p>Description: User can perform task with a brief touch.</p> <p>Example: Switching lights, clicking, pressing buttons.</p>	
---	--

<p>9. Audio Notification PU</p> <p>Description: User is being notified by some sounds regarding certain events.</p> <p>Example: time-over, error, collision-avoidance.</p>	
---	--

<p>10. Navigable View PU</p> <p>Description: User can view and navigate and zoom.</p> <p>Example: PDF docs, Maps, Virtual Environments.</p>	
--	--

<p>11. Live View PU</p> <p>Description: User can view live events from different angles.</p> <p>Example: rear-view cams, view-finding, mirrors.</p>	
--	---

<p>12. Multimedia PU</p> <p>Description: User can watch, listen and/or interact with multimedia contents</p> <p>Example: Video/audio playing, sound recording.</p>	
---	---

Appendix C

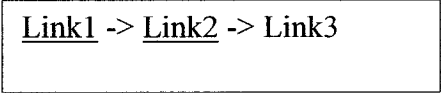
List of Suggested Prototype Elements (PEs)

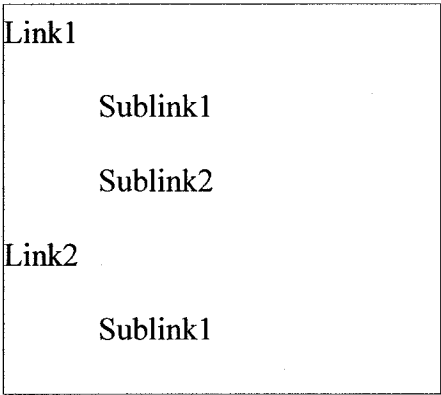
C.1 A Note about Prototype Elements

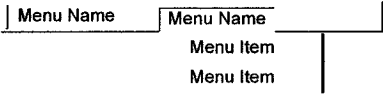
In this thesis, we present elements that should be arranged on prototypes. The prototype generation process is based on Presentation Units that are listed in Appendix B. These elements cover most of the interface components in common desktop and web-applications, as well as, widgets found in vehicle. We argue that this list can be expanded to include other elements corresponding to some other kinds of applications too. In that case, the matrix presented in chapter 5 of this thesis will be extended and new prototypes of innovative systems could be generated.


C.2 Prototype Elements

1. Tools Collection PE

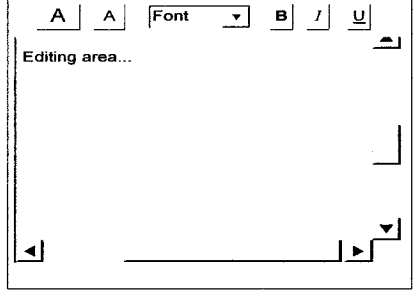
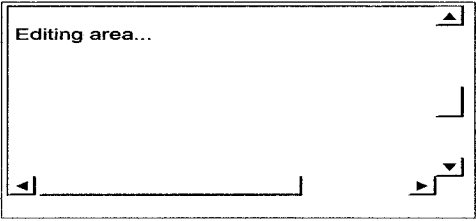
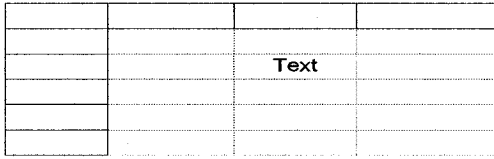
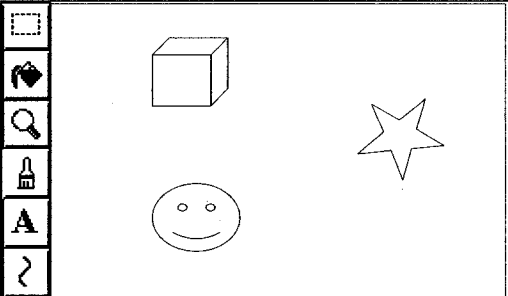
<p>PE1.1 Navigational Path (series of hyperlinks)</p> <p>Description: Typically used in web-applications to show the navigational path followed. Each word is link and could be clicked to go to corresponding page.</p>	 <p><u>Link1</u> -> <u>Link2</u> -> Link3</p>
---	---

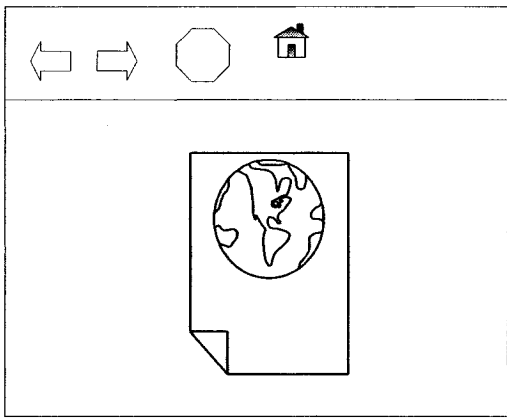
<p>PE1.2 Navigational hyperlinks</p> <p>Description: Typically used in web-applications to show the structure of navigation. Each word is link and could be clicked to go to corresponding page.</p>	 <p>The diagram shows a vertical list of text elements: 'Link1' at the top, followed by 'Sublink1', 'Sublink2', 'Link2', and 'Sublink1' at the bottom. All elements are aligned to the left within a rectangular box.</p>
---	---

<p>PE1.3 Menu items in Menu bar.</p> <p>Description: Different features (or tools) are grouped and each group is accessible under its common name.</p>	 <p>The diagram illustrates a menu bar structure. It shows a horizontal line with a box labeled 'Menu Name' on the left. A vertical line descends from the right side of this box, and another horizontal line extends to the right from the top of this vertical line, forming a T-shape. Below this horizontal line, the text 'Menu Item' is written twice, once on each side of the vertical stem.</p>
---	--

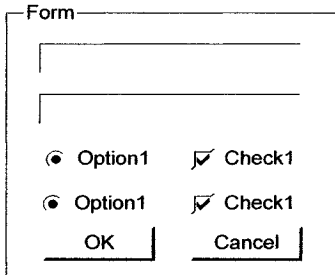
<p>PE1.4 Toolbar for collection of any tools.</p> <p>Description: A set of common tools are represented with a picture and displayed for easy access. Selecting a picture (by clicking or pointing) enables that tool or feature.</p>	 <p>The diagram shows a horizontal rectangular box containing three small icons: a document icon, a trash can icon, and a save icon.</p>
--	--

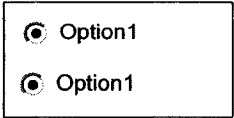
2. Linear View PE

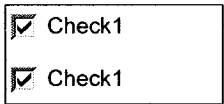
<p>PE2.1 Enhanced text editor</p> <p>Description: Complex component that allows entering and editing, as well as changing the attributes of the text.</p>	
<p>PE2.2 Simple text-editor</p> <p>Description: Enables entering and editing text. Changing attributes of text is not possible.</p>	
<p>PE2.3 Grid view and editor</p> <p>Description: Enables entering and editing text in a grid/table.</p>	
<p>PE2.4 Graphical editor</p> <p>Description: Enables drawing and editing graphical shapes.</p>	

<p>PE2.5 Webpage</p> <p>Description: Enables viewing contents of World-Wide Web. Editing the contents is not possible.</p>	
---	--

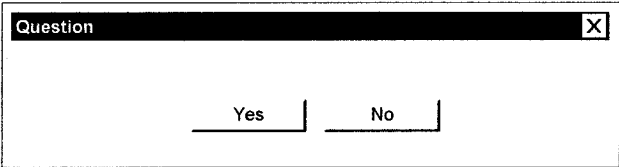
3. Form PE

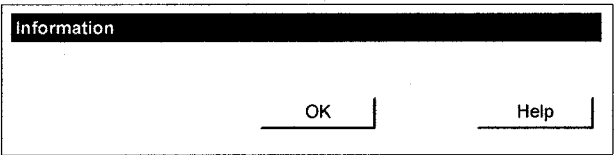
<p>PE3.1 Form with different types of fields</p> <p>Description: Enables entering and editing values in a form containing multiple types of fields. There are always buttons to submit the information and to clear the form.</p>	
--	--

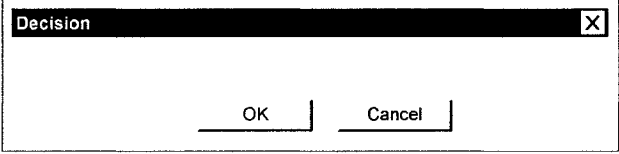
<p>PE3.2 Form to select from mutually exclusive options.</p> <p>Description: Enables changing value of an attribute. Different options are listed and user can choose only one.</p>	
--	---

<p>PE3.3 Form to select multiple options.</p> <p>Description: Enables changing value of an attribute. Different options are listed and user can choose multiple options.</p>	
---	---

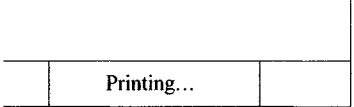
4. System-Dialog PE

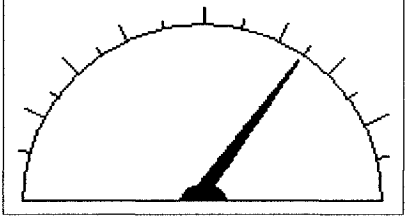
<p>PE4.1 Question Dialog</p> <p>Description: System asks user a question. User can respond in Yes or No.</p>	
---	---

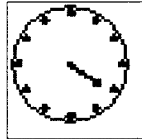
<p>PE4.2 Information Dialog</p> <p>Description: System informs user of some event. User presses OK button to acknowledge the information or press Help button to know more about the information.</p>	
--	--


<p>PE4.3 Decision Dialog</p> <p>Description: System asks user to take a decision to take an action.</p> <p>User can signal to go ahead or cancel the action.</p>	
---	--


5. Indicator PE

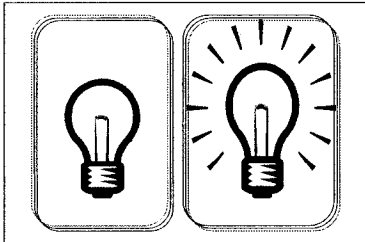
<p>PE5.1 Internal information in status bar.</p> <p>Description: System indicates an action being performed. No action is required on user's part.</p>	
---	---

<p>PE5.2 Widget for analog data</p> <p>Description: Indicates changing analog data.</p>	
--	--

<p>PE5.3 Widget for analog data</p> <p>Description: Indicates changing analog data.</p>	
--	---

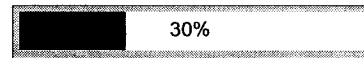
<p>PE5.4 Widget for analog data</p> <p>Description: Indicates changing analog data. The date is composite of two variables.</p>	
--	---

<p>PE5.5 Widget for numeric data.</p> <p>Description: Indicates changing numeric data. Need special displays (like Liquid Crystal) to show.</p>	
--	---

<p>PE5.6 Widget for Boolean data</p> <p>Description: Indicates Boolean data. The two figures below shows the two states; one on left is “0 (zero)”/”Off”... and the one on right is “1”/”On”...</p>	
--	--

PE5.7 Widget for Progress

Description: Indicates progress of an action system is performing. The level of colored/shaded bar inside the rectangle shows the progress made so far and the remaining rectangle indicates the time/amount of task yet to be performed. A number in the middle show the same information in term of percentage. The colored/shaded bar moves from left to right.






PE5.8 Widget for progress

Description: Shows the progress of an action, as explained above. The difference is the colored bar moves from down to top of the rectangle.



6. Value Changer PE

<p>PE6.1 Change value using keypad</p> <p>Description: Allows changing value of an attribute of an object using keyboard/keypad.</p>	
<p>PE6.2 Change value by rolling spin buttons</p> <p>Description: Allows changing value of an attribute of an object by going through many provided values. The value shown in the rectangle is the one most likely to be selected or the close in range of the value likely to be selected.</p>	
<p>PE6.3 Change value by selecting from list.</p> <p>Description: Allows changing value by selecting from a list. The values in the list are sorted by some logical order.</p>	

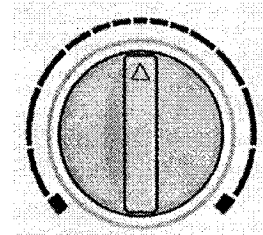
PE6.4 Change value by sliding along a line

Description: Allows changing value by sliding a handle along a line. The line is marked to show possible steps in the change. The more marks on the line, the more different values can be selected.



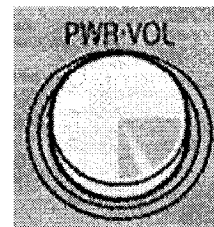
PE6.5 Change value by turning a knob

Description: Allows changing value by turning a knob. The marks around the knob show discrete values that could be selected. Turning the knob gives haptic feedback of discrete values.



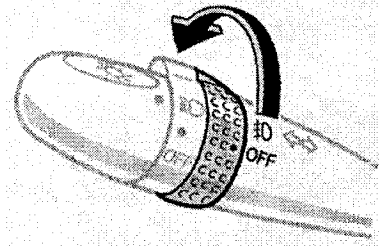
PE6.6 Change value by turning a knob

Description: Allows changing value by turning a knob. The values are continuous and there is no feedback of discontinuity when the knob is turned.



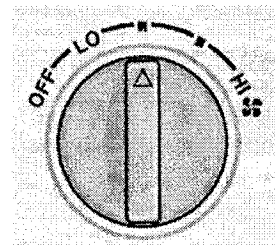
PE6.7 Change value by turning a knob

Description: Allows changing value by turning a ring on an arm/lever. The ring is integrated in the arm but its function is independent and different (but related) from the function of arm. Turning the ring gives haptic feedback of discrete values.



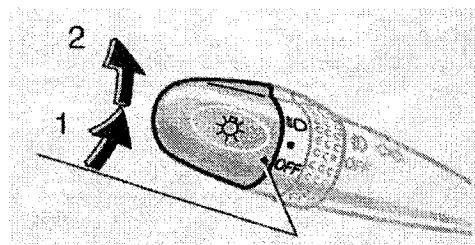
PE6.8 Change value by turning a knob

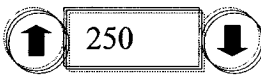
Description: Allows changing value by turning a knob. The marks around the knob show discrete values that could be selected. The range of values to be selected from is short so very few values could be selected. Turning the knob gives haptic feedback of discrete values.

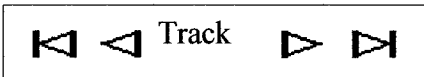


PE6.9 Change value by turning arm up/down

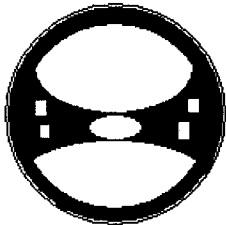
Description: Allows change value by pushing the lever/arm up or down. The possible states are usually limited to 3 or 4, which includes the Off/idle state.



<p>PE6.10 Change one value up/down</p> <p>Description: Allows changing values one by one by pressing the up and down buttons. This kind of gadget is suitable when scanning through a range of values. Pressing an up or down button for more than 2 seconds changes the value faster.</p>	
---	---

<p>PE6.11 Change values back n forth</p> <p>Description: Allows going back and forth in a range of values. The first and last values in the range are likely to be reached by user, so the buttons of reaching first and last values directly are especially given.</p>	
--	--

7. Constant Interaction PE

<p>PE7.1 Turn by turning a wheel</p> <p>Description: Allows user to change direction of an object. The turn of the wheel will make corresponding change in the direction of the object.</p>	
--	---

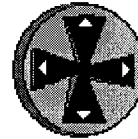
PE7.2 Turn by moving a stick

Description: Allows user to change direction of an object by moving a stick in different directions.



PE7.3 Turn by pressing left, right, up, down buttons

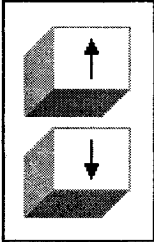
Description: Allows changing direction by pressing buttons in one of four directions. Pressing adjacent buttons simultaneously changing the direction diagonally.



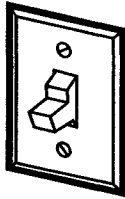
PE7.4 Increase / decrease using foot

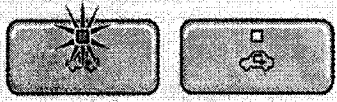
Description: Lets user increase or decrease speed (or other value) by applying pressure by foot on a pedal. Releasing pressure lifts the pedal back to up and decreases the speed (or other value). The neutral condition is the pedal lifted up when no pressure is applied over it.

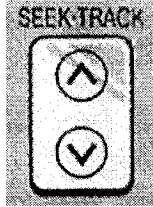


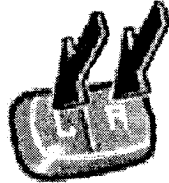
<p>PE7.5 Increase / decrease using keyboard arrow-keys</p> <p>Description: Lets user increase or decrease a value by pressing up and down arrow keys. This can also be used to move ahead/backward in some cases.</p>	
--	---


8. One-touch Interaction PE

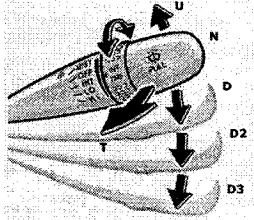
<p>PE8.1 Toggle by turning up/down</p> <p>Description: Lets user change Boolean value, like On/Off etc. Pulling the small lever up or down toggles the value.</p>	
--	---

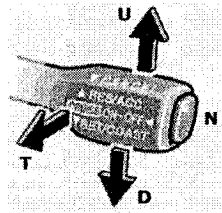
<p>PE8.2 Toggle by pressing same button</p> <p>Description: Lets user change Boolean value, like On/Off etc. Pushing the button once toggles the value. Button can have a light to show the current state.</p>	
---	--

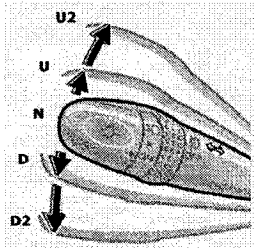
<p>PE8.3 Press button edges to change</p> <p>Description: Lets user change values by pressing up or down edge of a rectangular button. The values are changes one by one.</p>	
--	---

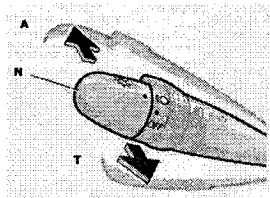
<p>PE8.4 Turn/move two ways</p> <p>Description: Allows turn or move two ways by pressing wither edge of a two-way (three state) button.</p>	
--	---

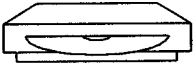
<p>PE8.5 Turn/more four ways</p> <p>Description: Allows turning or moving four-ways using a four-way button.</p>	
---	---

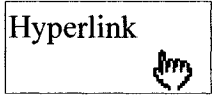
<p>PE8.6 Pull / Push arm</p> <p>Description: Change state of an object (or change value) by pulling arm gradually down.</p>	
--	---

<p>PE8.7 Pull / Push arm</p> <p>Description: Change state of an object (or change value) by pulling arm in three different directions.</p>	
---	---


<p>PE8.8 Pull / Push arm</p> <p>Description: Change state of an object (or change value) by pulling arm gradually up or down.</p>	
--	---

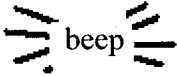
<p>PE8.9 Pull / Push arm</p> <p>Description: Change state of an object (or change value) by pulling arm towards or away.</p>	
---	---

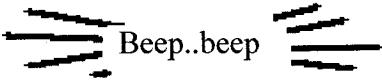
<p>PE8.10 Insert in a slot</p> <p>Description: Allows user to insert an object, like CD or cassette, into a slot. The object needs to be pushed once into the slot and the slot accepts the object.</p>	
--	---


<p>PE8.11 Click on hyperlink</p> <p>Description: Lets user jump to a certain page but clicking once on a text using a pointing device like mouse. The text itself serves as a link.</p>	
--	---


9. Audio Notification PE


<p>PE9.1 Give a voice command</p> <p>Description: User can give voice-command to the system.</p>	
---	---

<p>PE9.2 An alert</p> <p>Description: User is alerted of an event with a brief sound.</p>	
--	---

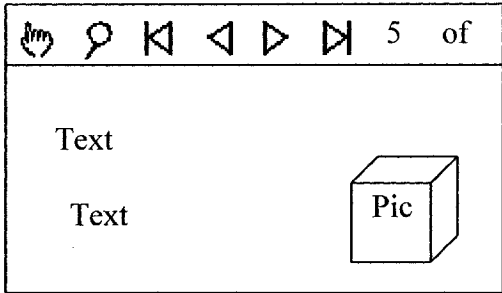
<p>PE9.3 A warning</p> <p>Description: User is warned of some problem with repeated beeps.</p>	
---	--

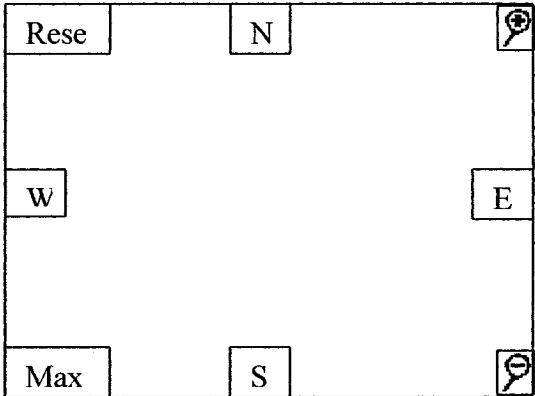
<p>PE9.4 Danger</p> <p>Description: User is alerted of a serious event with a loud sound.</p>	
--	---

<p>PE9.5 Feedback</p> <p>Description: User is given feedback of the result of an action performed by system (possibly triggered by user). This brief sound is different from that of an alert.</p>	
---	--


<p>PE9.6 Greeting</p> <p>Description: User is greeted at the start or end of the system or upon completion of a complex task.</p>	
--	---


10. Navigable View PE

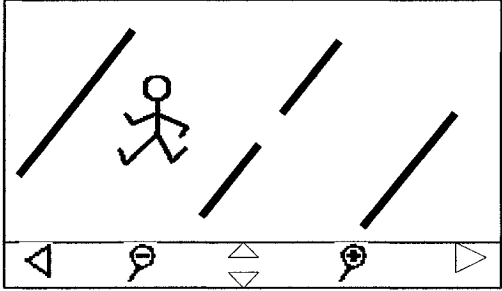
<p>PE10.1 Navigate in Textual artifact</p> <p>Description: Lets user navigate in a textual artifact. This includes going back and forth between pages and zooming-in and -out on the artifact.</p>	
---	--

<p>PE10.2 Navigate in geographical artifact</p> <p>Description: Lets user navigate in a geographical artifact, like a map or floor plan. This allows viewing in North/South/East/West directions and zooming-in and -out on the artifact.</p>	
--	---

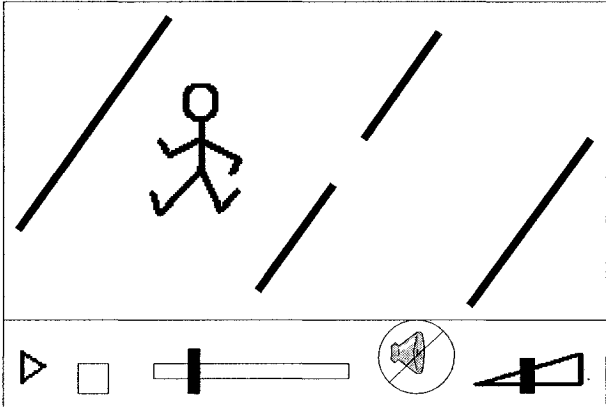
11. Live View PE

<p>PE11.1 Rear view</p> <p>Description: Lets user view objects that are behind.</p>	
--	---

<p>PE11.2 Side view</p> <p>Description: Lets user view objects that are behind sideways.</p>	
---	---

<p>PE11.3 View anywhere</p> <p>Description: Lets user view object that are in proximity, in any direction. It is a screen projecting view through a camera which could be controlled (turned or zoomed).</p>	
---	--

12. Multimedia PE

<p>PE12.1 Watch and listen a video</p> <p>Description: Lets user view video along with associated audio. The video/audio can be controlled (stopped, paused, resumed, forwarded, muted etc.)</p>	
---	--

PE12.2 Watch listen and interact with video

Description: Lets user experience video/audio. User can also interact with the video directly on the screen (using touch-screen technologies).

