UNIVERSITY OF Southampton

Faculty of Physical Sciences and Engineering

Electronics and Computer Science

Framework for Improved Capture of Usability Requirements through Usage-Centred Design

by

Kholod Jeza Alotaibi

A thesis submitted in partial fulfillment of the requirement for the degree of Doctor of Philosophy in Computer Science at the University of Southampton

January 2015

University of Southampton

ABSTRACT

Faculty of Physical Sciences and Engineering

Electronics and Computer Science

Doctor of Philosophy

Framework for Improved Capture of Usability Requirements through Usage-Centred Design

Kholod Jeza Alotaibi

This research makes an in-depth examination and comparison of plan-driven and agile methods in software design to justify the need for a more balanced approach that combines the controlling and disciplining capabilities of the former with the flexibility of the latter. It also examines the concept of software usability and the process of gathering requirements in software engineering. It is noted that pure agile methodologies tend to ignore usability for end users, hence the attention to User-Centred Design (UrCD). Software development methods based on UrCD attempt to alleviate this neglect, but they too are not sufficiently grounded in traditional plan-driven approaches that can adequately target usability concerns, especially during the planning phase of development. Capturing usability requirements could be key to ensuring a high degree of software usability for end users. It is suggested that Usage-Centred Design (UgCD), with its focus on actual software usage rather than users per se, can satisfy this need for greater software usability through increasing the likelihood of achieving a more complete and precise capture of usability requirements. This methodology introduced by Constantine combines task modelling as its plan-driven phase for identifying users, roles and essential tasks with an agile iterative component for facilitating continuous consultation with users to also enable refinements to be made to the requirements. The focus in this study is on the requirements analysis phase of UgCD to demonstrate how it can be applied for better capturing usability requirements in terms of completeness and preciseness. The usability aspects analysed are learnability, rememberability, efficiency, reliability, and user satisfaction. Research was carried out involving a survey, interviews and a demonstration of UgCD at a higher education institution in Saudi Arabia during a revision of an e-learning software, and a further follow on survey and interview. The survey investigated existing usability requirements gathering practices, and the current methods used for the sake of ensuring and thereafter testing for usability. The interviews provided insight into these developer practices to especially ascertain how well they are able to capture usability requirements for particular aspects of usability and their awareness of these aspects. A total of 212 software developers working in higher education institutions throughout the Saudi kingdom participated in the survey, 20 of them participated further in the interviews from three different cities, and the UgCD demonstration was held at one higher education institution. The survey analysis included frequency, factor and correlational analyses, and the interviewee responses were analysed using thematic analysis.

The list of usability requirements captured during the UgCD demonstration was compared to the original capture at the time of construction and to the pre-UgCD period when usability questionnaires were being used. The UgCD implementation enabled a 7.5-fold increase in completeness and 43% improvement in preciseness compared with the original capture, and a 2.5 gain in completeness and 24% improvement in preciseness compared to the requirements captured using usability questionnaires. The pre-UgCD survey results and interview findings are also presented and discussed that show a need to promote not only UgCD, but also to raise awareness of usability itself and make developers appreciate its importance. This led to developing a framework of principles for implementing UgCD, which is presented to guide developers in capturing usability requirements for enhanced software usability. However, it is noted that in line with the need to understand and promote usability, developers also need to be encouraged to consult frequently with their potential and existing users and become accustomed to holding frequent meetings with them. These changes would be necessary in order to be able to implement UgCD fully. Regardless, the potential of UgCD is established at least with respect to its capability to make developers and users focus on usability and in capturing a more complete and precise set of usability requirements that pertain to a wide range of usability aspects. Also, the post-UgCD survey with the meeting participants and interview with the leading developer give further positive indications for the success of the UgCD trial, and its potential for being used to enhance software usability.

Key terms: requirements, requirements capture, usability, usability requirements, usage-centred design

Table of Contents

List of Tables and Figures

List of Tables

List of Figures

Declaration of authorship

I, Kholod Jeza Alotaibi, declare that this thesis bearing the title:

Framework for Improved Capture of Usability Requirements through Usage-Centred Design

and the work presented herein is entirely my own, and has been generated by me as the outcome of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as stated in the section on Published Works:

Signed:

Date:

Acknowledgements

This work is dedicated to all people who have helped me and supported me while this research was being conducted. In particular, it is my pleasure to thank my supervisor Dr Andy Gravell for his patience, and continual support, commitment and excellent supervision. I also thank my second supervisor Dr Lester Gilbert for his useful suggestions and guidance. In addition, I would like to thank my dearest parents; my mother, father and husband, for their unconditional love, support and encouragement. There are not enough words to describe how grateful, indebted and thankful I am. Without their constant moral encouragement, I could not have completed this project.

Published Works

The researcher participated in the 5th Saudi International Conference 2011 (SIC05) in connection with her study, which was held in Coventry, UK from the 23rd to 26th of June, 2011. The SIC2011 was hosted by WMG, The University of Warwick, and the poster was titled: '**The Management of Web Development Projects in Saudi Arabia and Promoting Agile Methods**' (Number: 112; Category: ICT (Paper)

The researcher also participated in the International Conference on Software Engineering and Applications (ICSEA 2013) that was held in Dubai, UAE on 2-3 December, 2013. A paper was submitted that contains substantial portions of this report, and was published in a Special Journal Issue of the journal **Advances in Software Engineering and Applications**.

A paper titled '**Requirements gathering for improved software usability and the potential for Usage-Centred Design**' was presented at the World Academy of Science, Engineering and Technology's 12th International Conference on Computer Science and Software Engineering (ICCSSE) that was held in London, UK on 20-21 January, 2014. The reference for this paper is:

> Alotaibi, K.; Gravell, A. (2014), 'Requirements Gathering for Improved Software Usability and the Potential for Usage-Centred Design', World Academy of Science, Engineering and Technology, International Science Index 85, *International Journal of Computer, Information Systems and Control Engineering*, 8(1), 58-63.

The researcher will also be participating in and presenting the results and findings of her study at the forthcoming 13th International Conference on Software Technology and Engineering (ICSTE 2015), to be held in Jeddah, Saudi Arabia on 26-27 January, 2015.

Another paper titled '**Challenges in Promoting Software Usability & Applying Principles of Usage-Centred Design in Saudi Arabia**' has been submitted for presentation at the 13th International Conference on Software Engineering and Applications, to be held in Dubai, UAE on 26-27 February, 2015.

Abbreviations used

ACD	Activity Centred Design
ASM	Abstract State Machine
AUP	Agile Unified Process
CASE	Computer Aided Software Engineering
CCT	Concur Task Tree
CDR	Central Design Record
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CUSQ	Computer Usability Satisfaction Questionnaire
DSDM	Dynamic Systems Development Method
HCD	Human Centred Design
HCI	Human Computer Interaction/Interface
ISD	Information Systems Development
ISDM	Information Systems Development Method
JAD	Joint Application Design
JITR	Just-In-Time Requirements
JRP	Joint Requirements Planning
LOTOS	Language of Temporal Ordering Specification
MBASE	Model-Based System Architecting and Software Engineering
OOA	Object Oriented Analysis
OOD	Object Oriented Design/Development
PAF	Principal Axis Factoring
PCA	Principal Component Analysis
PD	Participatory Design
QUIS	Questionnaire for User Interaction Satisfaction
RAD	Rapid Application Development
RAS	Requirements Analysis Sheet
RE	Requirements Engineering
RSD	Requirement Specification Document
RUP	Rational Unified Process
SALUTA	Scenario-based Architecture Level UsabiliTy Assessment
SBD	Scenario Based Design

SDLC	Software Development Life Cycle
SPC	Statistical Process Control
SPEM	Software Process Engineering Metamodel
SPI	Software Process Improvement
SRS	Software Requirement Specification
SUMI	Software Usability Measurement Inventory
SUS	System Usability Scale
TAM	Technology Acceptance Model
UCSD	User-Centred Systems Design
UML	Unified Modelling Language
UgCD/U-CD	Usage Centred Design
UrCD/UCD	User Centred Design
VBSE	Value Based Software Engineering
WAMMI	Website Analysis and Measurement Inventory
XSBD	eXtreme Scenario-based Design

Note: The abbreviations UgCD and UrCD were devised to distinguish between the two.

Chapter One

Introduction

Chapter 1: Introduction

1.1 Overview of the dissertation

This chapter of the dissertation briefly introduces UgCD (Usage-Centred Design), which is a usability focused software development methodology and is the central focus of this paper as applied for capturing usability requirements. This approach to software development is defined in 1.7 and detailed in chapter 3. The problem statement establishes the importance of early gathering of requirements and ensuring software usability. Section 1.4 specifies the aim of the study, and the main and sub-research objectives, which then leads to the formulation of the research questions in section 1.5. The significance of the study is outlined in 1.6 and section 1.7 defines the most important terms constantly referred to in this paper.

The second chapter provides an in-depth examination of plan-driven and agile methods from a comparative perspective, which includes justifying the need for a balanced approach, exploring some existing combined plan-driven and agile methods, and outlining the necessary ingredients for a balanced mix of development practices. Chapter three details Usage-Centred Design (UgCD) as mentioned, which includes contrasting it with general User-Centred Design methods and an attempt at showing how it can be advantageous for addressing usability concerns. The concept of usability is then explored in greater detail in chapter four beginning with a discussion of software quality and first establishing the importance of usability. Chapter five refocuses the paper on the process under concern in this study, which is that of requirements gathering and capturing. This includes justifying the importance of gathering requirements, providing an overview of the field of Requirements Engineering, describing different types of requirements, and detailing how requirements are actually gathered. Different models devised to gather requirements are also explored prior to detailing how requirements would be gathered under UgCD.

Chapter 6 details the methodology that was applied in this research to find out how well usability requirements are being captured at present, which involved a survey and interviews of developers. The samples obtained are described in chapter 7. The survey results are reported in chapter 8, and the interview findings in chapter 9. The UgCD methodology was then demonstrated to show how it can be used to capture requirements in

a more complete and precise manner. The outcome of this trial is detailed in chapter 10, and the post-UgCD trial impact in chapter 11. A concluding discussion follows in the final chapter 12 in which the framework presented earlier for ensuring usability focused development while capturing requirements is recommended along with other related recommendations.

1.2 Usability issues in software design

Usability is a very important concept in software design in this age when people are giving more attention to the human side of computing and finding ways to enhance the user experience. The problem however, is that usability is difficult to define to begin with, which is evident from the various definitions offered, including standard specifications and the fact that definitions have evolved over time (see 4.2 Introduction to usability). Regardless, usability has become an established concept in software engineering, and is accepted as a basic quality related to the productivity and acceptance of software (Abran et al., 2003).

Another related issue is in measuring usability. In spite of its importance, measuring usability can often be frustrating for developers, and even if the procedures are clear, it is usually time consuming and expensive. The difficulty in measuring usability is due to it being an emergent rather than a specific property, given its dependence on interactions between users, tasks, products and environments (Hertzum, 2010). This also assumes that developers want to measure usability and have a clear idea of what usability is because, as shown by Alotaibi (2013), there is a lack of awareness of the concept of usability and usability engineering practices in general.

Where usability is accounted for during development, the practice is usually handled by adding a component targeted at ensuring usability to an existing development process. Otherwise, incorporating user-centred or other usability focused techniques does not appear to be a common practice. The motivation for this study is precisely this observation that usability appears to be neglected by software development teams in Saudi Arabia, and in Saudi Arabia in particular, it seems based on personal observations that understanding of the concept is either absent or vague at best (see interview findings in 9.2.2 Responses to question 2). Moreover, a need was perceived for promoting a usability focused method rather than making an adjunct to another development process.

Agile software development processes have existed and have been promoted for several years now but questions of their suitability have only been based on anecdotal accounts of people's experiences (Turk et al., 2005). Agile methods are based on core assumptions that may not be valid in all situations and environments. These assumptions include availability of customer teams for a frequent and constant interaction with developers, ability to structure projects for short iterations, that documentation is counter-productive, that continuous redesign is possible, that there is no dramatic increase in the cost of change, etc. For this reason, a new combined traditional-agile method has been proposed.

In particular, Kavitha & Thomas (2011) identified inadequacies in how to handle requirements, although they claim SCRUM can adequately deal with ascertaining user requirements by having an on-site customer. But this could be too much to expect from the users, and requirements can also change. Also, it may be more important, especially in the early stages, to give priority to the functional requirements of a software over non-functional requirements, such as aesthetic requirements.

Key features of the new approach will be requirements gathering with a focus on developing a 'usage' rather than a user centred design. Existing literature on the use of agile or hybrid agile methods for e-learning focus largely on user-centric aspects such as user satisfaction and ease of use. They lose sight of the main purpose of the software; on the characteristics that make it usable. In fact, there is inadequate integration of usability and agile methods, there are usability issues in software developed using purely agile methods; there is focus on user aspects such as motivation, appeal (aesthetics), user satisfaction, etc. but not on the more important goal of actually using the software according to the purpose for which it is developed.

1.3 Introduction to UgCD

1.3.1 Key features

UgCD (Usage-Centred Design) combines elements of both traditional and agile methods. Specifically, it incorporates a model-driven phase in which requirements gathering is an important component, into an agile framework. Its key features are therefore gathering user requirements (as in JAD) and devising models for planning. The focus throughout is on usage, that is, on the tasks that users will have to carry out and on the way these tasks are accomplished (Constantine et al., 2003). It has the aim of trying to make the users of the

software use the software more efficiently and effectively by enabling users to satisfactorily accomplish essential tasks, especially in terms of enhanced performance and ease of learning, and thereby ensuring its usability.

UgCD thus differs from UrCD (User-Centred Design) in that the focus is on usage rather than on users per se, hence its description of being a usage-centred rather than a user-centred approach to software design and development. Specifically, UgCD is distinguished by its attention to tasks, and in ensuring all the requirements for each task are met. Capturing the requirements for these tasks in turn ensures high software usability because all the requirements are concerned with using the software and getting it to do what the users want it to be able to do. It can therefore be perceived to be better in the sense that there are fewer, if any, chances of missing any essential requirements as far as ensuring usability is concerned.

1.3.2 UgCD and usability

Among other software quality attributes, UgCD is believed to enhance software usability (Constantine & Lockwood, 2002b; Constantine, 2002; Barksdale & McCrickard, 2012), and has also been shown to do so (Dabbagh, 2012). However, no research has been found that has specifically examined the role of the requirements gathering/capturing process in enhancing software usability under UgCD. In order to demonstrate this potential enhanced usability, the existing requirements capturing process will be examined first to see how well they are able to ensure usability. Focus in this study will be on the five attributes of usability deemed to be most important for e-learning software, based on the literature review, namely (1) learnability, (2) rememberability, (3) reliability, (4) efficiency, and (5) user satisfaction. In the later experiment phase, an attempt will be made to show how UgCD (based on a model-driven and agile combined approach) can enhance usability.

1.4 Goal, aims and objectives

The aim of this research is to show how UgCD can be used to improve the capture of usability requirements and to then develop a framework to guide software developers to do the same for the ultimate objective of improving the usability of the software. This usage-centred approach is being suggested as opposed to a user-centred approach.

The original contribution will be a framework or set of principles devised to guide developers on how to better capture requirements for ultimately enhancing software

usability. Although this would be applicable to software generally, it will be demonstrated using e-learning software. A secondary contribution will be to study UgCD empirically and its potential for better capturing usability requirements.

The main research objective is as follows:

1. To test the suitability of a proposed framework for achieving improved capture of usability requirements through UgCD in terms of completeness and preciseness;

The sub-objectives of the research are:

2. To ascertain how usability requirements for developing software are captured at present;

3. To find out how well the existing requirements capturing process is satisfactorily able to capture usability requirements;

4. To show how UgCD can be used to improve the usability requirements capturing process for enhancing the usability of software through using e-learning software to demonstrate the process.

The objectives map onto the research as follows:

- The second objective was achieved through the pre-UgCD survey and interviews;
- The first, third and fourth objectives were achieved through carrying out the UgCD trial.

1.5 Research questions and hypotheses

The main research question for this study is as follows:

1. How suitable is the proposed framework for guiding developers in capturing usability requirements completely and precisely?

In order to address this research question, the following sub-questions have been framed:

2. How are usability requirements being captured at present?

3. How well is the existing requirement capturing processes at a selected institution able to capture usability requirements (that is not already using UgCD)?

4. Does UgCD enable the usability requirements to be captured more completely

and precisely?

The following two hypotheses have been framed to assist in answering the fourth sub-research question:

H1: The process for capturing requirements under UgCD yields a more complete, i.e. longer, list of requirements necessary for enhancing software usability compared to the existing process in each of the five selected usability categories* and in their sum total (after taking into account wholly/partially stated requirements and matching them).

H2: The process for capturing requirements under UgCD yields a more precise, i.e. clearly expressed, set of requirements necessary for enhancing software usability compared to the existing process in each of the five selected usability categories* and in their sum total (as rated by a panel of experts for each distinct aspect* of usability identified for completeness).

*The five selected usability categories are learnability, rememberability, efficiency in use, reliability in use, and user satisfaction, and the distinct aspects of usability to be identified will be sub-categories of these. See section 6.7.2 Adopted measures for further details.

1.6 Significance and potential contribution of study

The literature mostly mentions studies on applying agile methods in user-centred design contexts. Even methodologies applied to e-learning software development are mostly focused on user-centric factors. For e-learning software, UgCD would focus on the actual usage or learning aspect of the software, i.e. for promoting effective learning as the end goal. It is believed that the task focused nature of UgCD can help to ensure high software usability and that e-learning software can benefit from this approach. Unfortunately, there is a lack of studies on e-learning software usability (Srivastava et al., 2009), and usability of software developed using agile methods is questionable (Turk et al., 2005). Usability is a very important consideration for e-learning software as it helps to develop better systems with improved didactical and pedagogical approaches (Novak et al., 2010), but for this type of software, it is often neglected (Kruse, 2002). The main reason for this neglect is because usability is considered tedious, least rewarding and expensive to test (Mueller, 2009). There is no standard means to evaluate the usability of e-learning software (Feldstein (2002), for which reason, important usability aspects other than the level of user satisfaction tend to be ignored, hence the need for better understanding of usability with

respect to e-learning software.

In particular, there are inadequacies in the way in which requirements are handled (Kavitha & Thomas, 2011), so this research will also satisfy the growing interest in requirements engineering (Zhang, 2010). A framework or set of principles will be devised to guide developers on how to better capture requirements for enhancing software usability that could be applied to software generally but especially to e-learning software. A secondary contribution will be to study UgCD empirically and its potential for better capturing usability requirements.

1.7 Context of the study

The study context is centered firstly on e-learning software, and secondly on software developers in higher education institutions in Saudi Arabia.

1.7.1 Cultural context

Culture refers to the “collective programming of the mind that distinguishes one group of people from another” (Hofstede, 2001: 9). Saudi Arabia is distinguished by being a highly conservative and religious country. Its culture can be described as collectivist, relatively homogenous, hierarchical and patriarchal. For this kind of culture, Hofstede's dimension of 'uncertainty avoidance' and Hall's (1983) classification of 'high context' are of particular relevance.

Having a high context, as opposed to a low context, indicates the preferred communication styles. Arabs tend not be explicit in their communications. Instead, they prefer to embed the actual meanings, which is similar to although not to the same level of extremity as the Japanese style of communicating (Table 1). Listeners therefore have to guess the actual meanings, and communications with people from 'low context' cultures, who are accustomed to being open, precise and direct, tend to be challenging (Gudykunst & Ting-Toomey, 1989). Greater openness only comes after both sides first work on establishing trust.

Table 1: High and low context cultures

High context cultures
Japan
Arab countries
Greece
Spain
Italy
England
France
North America
Scandinavian countries
German-speaking countries
Low context cultures

Source: Hall & Hall (1990)

Saudi Arabia also has a collectivist society because of the emphasis on group needs and interests, and in Saudi culture, honour plays an important role and the social customs include strict gender segregation. With respect to software development, this kind of environment is restrictive. In particular, this cultural context can present difficulties in arranging for the kind of collaboration, open or face-to-face communication and social interaction that is expected when implementing agile methods, especially between members of the opposite gender. Team, communication and customer relationships are important cultural factors identified by Chan & Thong (2008). Also, the tendency to avoid uncertain situations means that developers in general would not be willing to try an alternative software design method to what they are already accustomed, which in the Saudi context means anything other than traditional plan-driven methods.

In an academic institutional environment, these aforementioned problems exist but are less pronounced compared to society generally because the management, teachers and students are used to working closely together and the needs of teachers and students is a top priority. Moreover, communication technologies can easily be used in supporting distributed agile practices through allowing both formal and informal controls necessary for collaboration (Persson, 2011). The core value of cultivation is also present because such organisations focus on future possibilities and appreciate students' learning and creative development. Nonetheless, organisational culture is affected by the prevailing societal culture at large. Taking this culture into account is important because it determines the extent to which agile principles can be accepted and implemented in an organisation accustomed to using

traditional methods (Chan & Thong, 2008).

In such an organisation, adopting an agile method or other software design methodology otherwise dependent on relating and interacting with users, open communication and collaboration, presents a major challenge. How that challenge is dealt with can affect the extent to which the methodology is applied and possibly the success of the software project. Tolfo & Waslzwich (2008) point out that organisations in cultures that are highly incompatible with the values and principles expected in agile methods are likely to experience exhaustion. This is likely due to difficulties experienced in overcoming certain inhibitions and restrictions imposed by the culture but required by agile methods.

A study by Kautz (2009) carried out a cultural analysis on three organisations that adopted agile methods for their systems and software development. The organisations were present in three different cultures, which were cooperation and competence based, and one was characterised as having a subculture within a culture. It was found that agile development can easily thrive in different cultural contexts as long as some important conditions are met because of their ability to compensate. Four necessary conditions were identified, which are control, competence, collaboration and cultivation, which must be present to a significant degree.

1.7.2 Academic and e-learning context

The academic context of the study means that the users of the software are teachers and students. It may be that neither can be expected to fulfil the role of a full-time on-site customer, but they may still be willing to offer suggestions, work collaboratively, and even make some contributions. Unlike with culture, this context does not pose a constraint on applying UgCD, as collaborative working is not likely to be uncommon. One possible issue in this context is that of 'requirements volatility', that is, major changes may be expected in project requirements. However, unlike traditional methods, which are not able to cope with this scenario well, agile methods can be used to address this issue effectively (Davey & Parker, 2010).

Using agile methods in supporting teaching/learning is referred to as the Agile Teaching/Learning Methodology (ALTM). It is designed to gather the best ideas and practices from software engineering and concepts from agile methods and applying them in delivering/taking courses (Chun, 2004). They have been shown to be especially useful to

aid teaching when combined with communication, creativity and enthusiasm (Tesar & Sieber, 2010). ALTM platforms utilise modern information and communication technologies to support collaboration and knowledge sharing, such as through instant messaging, commenting, wikis, blogging, etc.

1.8 Definitions of key terms

Definitions are given below of the key terms used in this study together with references to the later sections of this paper in which further details can be found.

Requirements

A statement of expected system characteristics or necessary attributes; 'A condition or capability needed by a user to solve a problem or achieve an objective; that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents' (IEEE Standard 610.12-1990); see section 5.1.1 for further details.

Requirements capture

A process typically undertaken during the system analysis stage, which usually involves elicitation for obtaining and possibly later refining requirements, and following which the elicited requirements are usually documented using either natural language or a conceptual model. See section 5.2.2 and 5.3. It can take the form of a formal process, a structured process, a social process, or a combination of these depending on the actual method used to capture the requirements of which there are many. Requirements Capture as distinct from Requirements Gathering, Requirements Engineering, Requirements Elicitation, etc. implies a sense of achievement in correctly identifying a requirement as is the focus of this study. This distinction is explained in detail in section 5.1.3.

Usability

Usability usually refers to how easy it is for a user to learn to operate, to prepare inputs for, and to interpret the outputs of a system (IEEE Standard 1061-1992). See section 4.2 Introduction to usability. Several components of usability have been identified, such as learnability, rememberability, effectiveness, reliability, satisfaction, etc. See section 4.7 and Table 5.

Usability requirements

Those requirements that directly impact on the usability of a software; Described by Bennett (2004: 121) as those requirements that ensure “a good match between the system that is developed and both the users of that system and the tasks that they will undertake when using it”. Such requirements prioritise usability and ensure usability goals are met (Hammond et al., 2004: 141). See section 5.2.2.

Usage-Centred Design

A model-driven software development methodology developed by Constantine & Lockwood that focuses on the tasks to be accomplished. As such, the software developed using UgCD is built around uses rather than on users per se, and due to this focus and distinction from UrCD, it prioritises usability throughout and thereby increases the chances of the software being highly usable (Juristo et al., 2003). UgCD involves exploratory task modelling during the design phase to identify roles and tasks, a procedure adapted from Jacobson's Use Case driven approach, and use cases are distinguished between those that are essential and those that are real. The JITR process is then applied to refine the requirements later. UgCD bears similarities with JAD and ACD. See sections 3.1 for description of UgCD, remainder of section 3 for other information related to UgCD, section 5.6 for details of how requirements are gathered under UgCD, and Figure 18 for the logical process of UgCD.

User-Centred Design

Software development methodologies that focus on software users, and which usually rely on scenarios. Differences with UgCD are discussed in 3.4 and summarised in Table 3.

1.9 Summary of chapter

The chapter introduces the study, which relates to the field of usability engineering in software design, specifically to the phase of requirements gathering. It identifies current software design practices as being dominated by User-Centred Design (UrCD) and agile methodologies, and introduces Usage-Centred Design (UgCD). UgCD combines elements of both traditional and agile methods by incorporating a model-driven phase into an agile framework. UgCD is implemented during the requirements gathering phase and is claimed to enhance software usability as an outcome of its capability to completely and precisely identify users, roles and tasks, and specially to gather usability requirements. It also provides for refining those requirements during its later iterative stages through allowing

continuous consultation with users. The need for focusing on usability is justified given the continued inadequacies in handling requirements (Kavitha & Thomas, 2011). The aim of the study is to show how UgCD can be used to improve the capture of usability requirements, and to then develop a framework of principles to guide software developers in capturing requirements for enhanced software usability. The need to combine plan-drive and agile methods, the concepts of UgCD and usability, and the process of requirements gathering are explored in depth in the subsequent chapters comprising of the literature review.

Chapter Two

Plan-Driven and Agile Development Methods: A Comparative Perspective

Chapter 2: Plan-driven and agile development methods: a comparative perspective

2.1 Comparative perspective of agile and plan-driven methods

A few methods from both traditional (plan-driven) and agile methodologies are compared and contrasted in order to highlight their respective advantages and disadvantages and to show how UgCD combines elements of both. The two can be considered as hard and soft methods respectively in the development of information systems. A trend can be observed that shows hard methods were prominent initially, but that they later gave rise to softer methods (see table below). This is also reflected on the timeline, which shows the emergence of various traditional and agile methods in Information Systems Development (ISD) over the years.

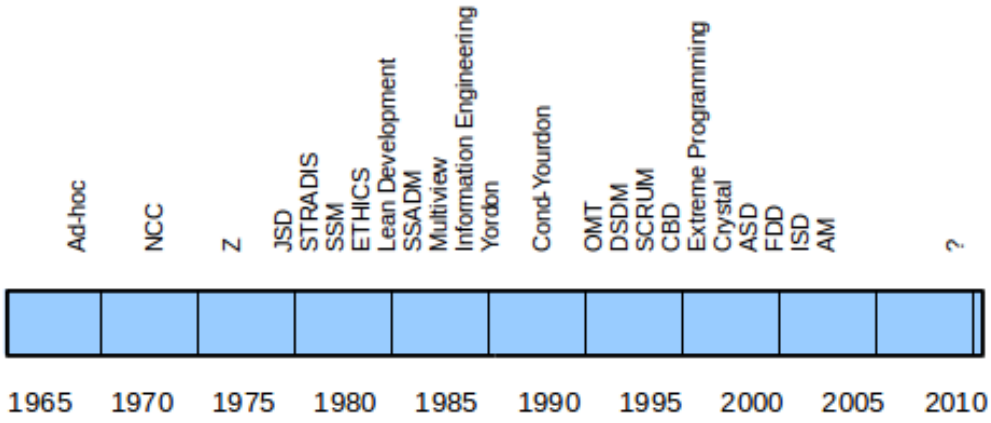
The focus of hard methodologies, which involve structured and organised plan-driven processes, is to make software quickly and in a way that is repeatable. The main success criterion is developer productivity. Soft methodologies at the other end aim to satisfy users. Hard methods deal mainly with issues such as structure, implementation and testability, whereas soft methods deal mainly with issues such as stakeholder communication, user participation and conflict analysis. This does not mean, for example, that hard methods do not involve users at all but if they are involved, their involvement is minimal, typically at the stages of feasibility analysis and final acceptance but not during the actual development.

Table 2: Prominence of hard/soft methods since 1960s

Period	1960s to early 1970s	Late 1960s to 1970s	Late 1970s to early 1980s	1980s to 1990s	Mid-1990s to present
Era	Pre-methodology era		Early methodology era (prescriptive)	Methodology era (prescriptive)	Post methodology era
Emphasis	-	Software product improvement (in emulation of traditional engineering methods)		Software process improvement (SPI) (through use of ISDMs)	Agile ISD – on points identified in the agile manifesto such as individuals and interactions
Methodologies	Ad-hoc approaches	(see timeline below)	SDLC-waterfall model	Unified process, object-oriented analysis, multiview, formal methods, etc.	Agile, scrum, crystal, XP, etc.
Related developments	-	Quality assurance practices such as SPC	Establishment of quality standards and guidelines		Kaizen, lean, six sigma, etc.

Sources: Stamelos & Sfetsos, 2007; Parsons & Lal, 2006 (adapted)

Figure 1: Forty years of ISD methodologies



Source: Stamelos & Sfetsos, 2007 (adapted)

2.2 Traditional methods: About CMM (Capability Maturity Model)

The use of CMM began with large-scale military software development, and it was promoted by the US Department of Defense through the Software Engineering Institute (SEI). The method was improved further to deal with the problem of failed projects. One of the guiding ideas behind CMM is that quality improves through gaining an increasing degree of control over processes (Orr, 2002). CMM has five levels, which are as follows: (1) Initial, (2) Repeatable, (3) Defined, (4) Managed, and (5) Optimising.

CMM tends to appeal to organisations in which managers want control and also to be able to grant certification. The latter gives an impression of credibility. CMM has earned a reputation of being a popular and well-documented method for software management. On the other hand, there is a concern that CMM is too focused on management processes instead of the quality of the end products. As for the certification, it is seen as marketing oriented and largely arbitrary as it relies more on paperwork and good documentation than actual management. Higher levels of certification require strong project management and are less concerned with production. This approach is not liked by many developers who would rather concentrate on the design and program instead of being required to complete a lot of paperwork.

Key points about CMM:

- Designed for greater control and to allow certification
- It is too focused on processes instead of the software
- It requires too much paperwork and documentation

2.3 Agile methods

Agile methods arose with the ‘object revolution’ in the late 1980s. OO techniques were then combined with features that now characterise agile methodologies. These include developing in short increments, working hand-in-hand, allowing for changes in requirements and minimal documentation. A comparative study on agile methods by Abrahamsson et al. (2003) identified four key characteristics that define all agile methods – agile methods are: (1) incremental, (2) cooperative, (3) straightforward and (4) adaptive. Their incremental nature leads to small releases with rapid release cycles; by cooperative is meant that developers and customers work together; being straightforward results in the

method being easy to learn and modify and well documented, and being adaptive makes it possible to make even last moment modifications. Agile methods also seek to eliminate waste and prevent miscommunication. The agile manifesto devised in 2001, encapsulates many of the common principles in agile methodologies. It mentioned focusing on individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan. Although the previous components (right) were continued to be valued, the replacements (left) were valued greater. This manifesto, restated below, clearly sets out the differences between agile and traditional methods.

The manifesto for agile software development:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Several principles can be identified that underlie the agile manifesto. Some of the key principles that characterise agile methods, and which make them distinguished from traditional methods, are listed below (adapted from Agile Manifesto and All About Agile). These are not universally agreed upon principles, and are presented in no particular order.

- The highest priority is to satisfy the customer through early and frequent delivery.
- There is active user involvement.
- Small, incremental releases are developed that are then iterated.
- Changing requirements are welcomed, even late in development.
- Face-to-face conversation is used to convey information.
- The development teams are self-organising.
- There is collaboration and cooperation between all stakeholders.

In practice, agile methods tend to be most suitable for smaller scale and stand-alone projects because of their less demanding nature and their ability to satisfy the need for

rapid development and to cope with continuous changes (Boehm & Turner, 2005).

Five major agile methods emerged in the 1990s: Crystal methods, Scrum, Dynamic Systems Development Method (DSDM), Feature Driven Development, and Extreme Programming (XP) (Rico et al., 2009: 7). These methods already existed at the time of the agile manifesto, which was formed to point out their commonalities and specify the four major underlying values along with 12 broad principles. These values are common to all the agile methods, and agile methods are essentially value-based.

Of the aforementioned methods, XP was the first to be described as 'agile', and its primary focus is on ensuring customer satisfaction and teamwork (Lankhorst, 2012: 19-20). Although XP's values make it agile (because it shares the values defined in the agile manifesto), XP differs fundamentally from other agile methods with respect to certain rules it follows. The few differentiating rules are continuous validation through testing, writing units before coding, and collective ownership (Maurer & Wells, 2003: 38). DSDM is also focused on iteration and incrementing, but in a more phased manner, and it emphasises risk mitigation, ensuring quality and measuring progress. DSDM is based on 8 principles that make it regarded as a 'heavyweight' variety of agile methods, and it arose from Rapid Application Development (RAD). In contrast, Scrum, which is a very popular method, is lightweight and relies heavily on meetings and collaboration. However, it is criticised by some for not being agile enough because it does not allow for adding new requirements during its time boxes called 'sprints' (Lankhorst, 2012: 20).

2.4 Object-oriented methods

Among the early software development methods were those described as 'Object-Oriented' in which a system is perceived as comprising of objects with there being relationships between them. OO systems provided advantages of greater control of data, re-usability, and better adaptability to changes despite there being some possible degradation of performance and usually high costs of training (Hevner, 2010). The various OO methods for software development then converged to produce the UML standard.

2.4.1 UML (Unified Modelling Language)

In addition to agile methods therefore, another major development in the field of software engineering of relevance to this study has been the adoption of UML (Unified Modelling Language) for describing object-oriented systems. UML has become the de facto standard

language used in Object Oriented Development (OOD) (Hudson, 2001). The UML standard, developed by the Object Management Group, details graphical notation techniques for creating visual models of software-intensive systems that are object-oriented. As such, UML is not in itself a software development method or programming language, and it is also not limited to specifying how diagrams are to be constructed; rather, it provides for a standard means to model these systems based on best practices and includes documentation techniques.

2.5 Comparison of traditional and agile methods

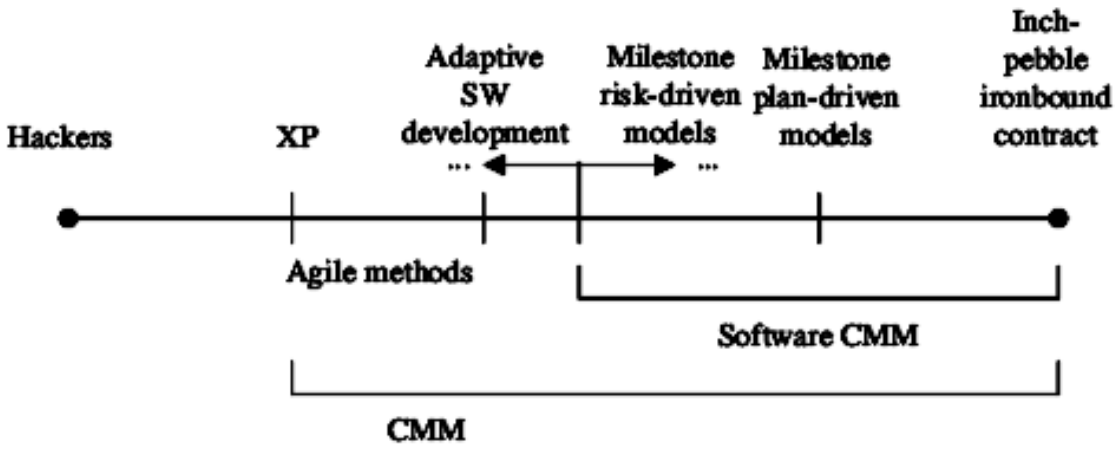
2.5.1 Planning and collaboration

In traditional methods, careful planning is given great importance and it is the responsibility of the project manager. These methods are often referred to as ‘plan-driven’, and they led to various prescriptive methods being developed. The prime focus was on quality and to ensure this, considerable time was spent in determining the requirements alone before coding for the software was even begun (Parsons & Lal, 2006). This typically involved feasibility studies and requirement validation, reviews and management, and the devising of class diagrams, data and object models, etc. This level of detailed planning was considered as essential whereas agile methods see this being done in advance to such a degree as time consuming and unnecessary.

Agile methods differ by putting much less responsibility on the manager and involving the technical people and users in the process to a greater or lesser degree. Planning does take place in agile methods but it is less extensive and the focus is on being efficient and effective. In XP for example, planning is suggested to cover only up to the end of the next iteration and longer term planning, if any, is limited (Beck, 2000: 85). The principle of ‘collective ownership’ in XP in which everyone is responsible for the entire system also reinforces the decentralist perspective of agile methods.

Boehm (2002) provided a useful illustration of the planning spectrum, which is reproduced in Figure 2 below. At the extreme ends are inch-pebble ironbound contracts and hackers and at the far end of each of these types are the plan-driven and agile methods respectively. It is to be noted that the centre of the spectrum that balances both extremes is occupied by risk-driven and adaptive methods. UgCD shares characteristics mostly in common with the somewhat planned but more adaptive agile methods.

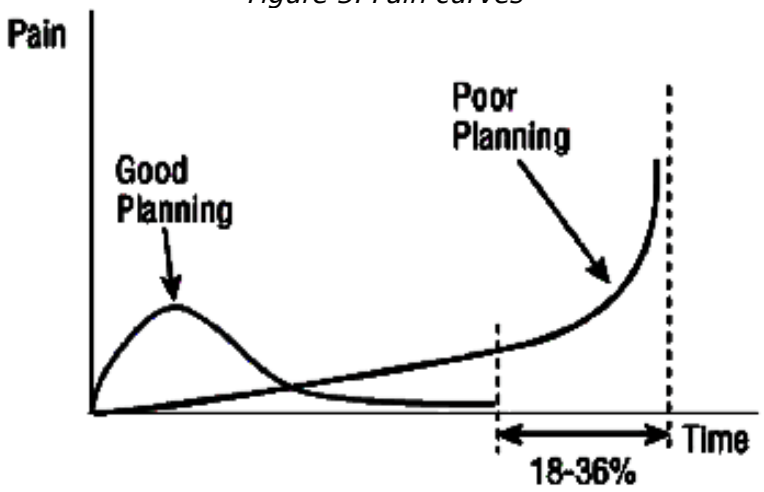
Figure 2: The planning spectrum



(Source: Boehm, 2002: 65)

To some extent, detailed planning in software development is essential because it allows for suitable scheduling, budgeting and quality control. It provides three potential advantages, namely (1) a reduction in uncertainty due to consideration of the likely outcomes and preparation of corrective measures, (2) an increase in understanding of the projects goals and objectives, and (3) an improvement in efficiency (Wysocki, 2011). Consequently, it is thought that a well-planned project is more likely to finish sooner, cost less and have better quality. Wysocki (2011) explains that proper planning is a painful' process but it pays off as the project progresses, and argues that by not planning, exposure to significant 'pain' is experienced later on. The effect of these two scenarios on time is illustrated by the graph below. Good planning has a higher initial pain but the project proceeds smoothly and on time whereas poor, little or no planning at the initial stage leads to an increasing amount of pain later and ultimately makes the project fail.

Figure 3: Pain curves



(Source: Wysocki, 2011)

This explanation is useful in contrasting between the effort made in terms of planning and its likely impact on the progression of the project, especially the time factor. It should also be pointed out that the poor planning scenario does not necessarily refer to an agile method having been adopted; rather, that planning has either not taken place at all or it has been done in an inadequate or ad-hoc manner. Consideration also needs to be made for justifying the planning and the extent to which it is advisable or required. This depends on the type of project, and on its size, complexity and degree of importance.

Traditional software methodologies required extensive planning because they were usually applied to large scale projects. Agile methods, which usually involve much less extensive planning, became popular because the initial 'pain' threshold was seen as too high and inappropriate for many smaller scale projects. Moreover, it is often the case that projects do not go according to plan and the plan needs to be constantly revised. This justifies the position of agile methods, which are typically characterised by short term planning and only provisional longer term planning. For many short projects, too detailed planning at the early stage is a sheer waste of time, especially if the project or any element involved is of a non-critical or non-complex nature. In such situations, spending too much time in planning results in a ‘high loss probability’ (P (L)) due to the time lost that could have been spent more usefully getting on with the project itself (Boehm, 2002). It could also prove to be a very expensive undertaking (Guerrero & Eterovic, 2004).

On the other hand, the involvement of users in agile methods is permitted to an extreme degree because it requires ‘on-site customers’ wherein a real customer is made to sit with

the team, answer their questions, set some priorities and even resolve their disputes (Beck, 2000: 60). This calls for a total full-time commitment that lasts throughout the duration of the project. It is therefore very demanding of the users to get deeply involved. A case study of an on-site customer in an XP based project confirmed this perception that a 100% available customer is demanding and also costly and difficult to arrange (Koskela & Abrahamsson, 2004). Furthermore, the empirical results showed that only 21% of the customer's effort was actually required by the software development team. They also pointed out that appointing an on-site customer could create a false sense of confidence. This is probably because the team is led to believe the on-site customer is representative of the potential population of users and will identify and resolve every issue that arises.

2.6 Justification for a balanced approach

There seems to be a greater acceptance now that neither traditional nor agile methods can be suitably applied in all software engineering contexts and that they each have their relative advantages and disadvantages. As pointed out by Parsons & Lal (2006: 9) for instance, it is accepted that although agile methods can be usefully applied in certain contexts, they are not suitable in all of them. In particular, agile methods tend to be inadequate in terms of being able to provide "a fully viable methodology" in cases where the organisation is complex and in which the individuals are not self-organising. This section explores issues that lead to justifying a balanced approach, and examples are given in 2.7 of two existing balanced approaches, namely the OSS model and SBD (Scenario-Based Design), and of recent hybrid models. Such approaches try to derive benefits of both discipline and agility. Further examples are given in 2.7.3 of more recent hybrid development models.

2.6.1 Adaptation and diversity

Sommerville (2007: vi) states that the basic processes involved, namely specification, design, development, verification, validation and management, have to be properly understood together with the techniques associated with them, to help decide the most appropriate method and how it can be adapted for the situation at hand. This is the right approach to take as it shows a better understanding of the nature of software development projects and is therefore better to satisfy their needs and requirements. It is too simplistic to consider a single fixed methodology, regardless of whether it is traditional or agile, as being able to deal with all software development projects.

It should be accepted that the methodology to be applied will be determined by the processes involved, and in particular, by the size and complexity of the project. Furthermore, it should be accepted that the methodology would need to be adapted according to the unique characteristics of the project, the working environment and available resources. Crystal agile methodologies do allow for adjusting the method according to the project's size and criticality but they do not advocate enough methods that can be usefully applied.

The point made by Sommerville (2007: 4) is that there is now a recognition of needing a diversity of approaches to software development in response to the diversity of different types of systems and organisations. Furthermore, despite acknowledging the differences, there is also a recognition of underlying fundamental notions common to all the techniques pertaining to process and organisation. He identifies four fundamental processes, some of which were mentioned above, which are (1) software specification, (2) software development, (3) software validation, and (4) software evolution. Specification is where the developers and users define what is to be produced as well as the constraints; development refers to the actual designing and programming; validation concerns software checking to ensure it is as required, and evolution concerns modifications in line with changing requirements. To illustrate the need for different development processes for different systems, he compares real-time software used in aircraft and e-commerce systems. Whereas in the former, it is necessary for specifications to precede development, in the latter case, it is usually acceptable for both to be developed together.

Agile methods do allow for adjustment in response to needs that emerge during the development process but they are all still designed to provide a universal solution instead of focusing on being situation appropriate (Abrahamsson et al., 2003). Moreover, they all cover only certain phases of the development life-cycle and they usually lack adequate support for project management. There is a need therefore, for greater methodological quality rather than for devising yet another universal method that attempts to satisfy every possible need. Beck did acknowledge that no single process can fit every software project (and stressed for satisfying individual requirements) but it is felt that agile methods went to the other extreme end of the spectrum from traditional methods and therefore failed to be more universally applicable and adaptable. The future direction should involve pursuing balanced combined methods with a better understanding of requirements, processes, resources, situational context, etc.

2.6.2 Issues with pure agile methods

One of the main concerns of agile methods is that as there is little planning beforehand and there is greater uncertainty instead. In particular, it proves to be difficult to estimate the costs of projects based on iterations (Stamelos & Sfetsos, 2007: 19). On the other hand, it could be said that agile methods save time and cost because of the quicker start of projects. Furthermore, it has been observed in agile development conferences that the "software process jacket" of many projects led by agile methods that cover enterprise architecture, patterns and software reuse, is getting heavier. This means it is becoming increasingly accepted that even if agile methods are used initially, it is not always possible to retain the same level of agility over time. One explanation could be that no matter how agile a team starts off, in the long term, it tends to settle on certain accepted patterns, techniques and habits, thus losing its agility.

Although the need for alternative approaches has been established, introducing agile methods in organisations that are accustomed to traditional/formal methods, i.e. entirely and with little if any preparation, is not likely to be satisfactory. This may not be due to deficiencies in the chosen agile method itself, but due to a number of possible changeover issues that may arise.

In general, smaller projects tend to be more suited for applying agile methods as they are better able to allow for rapid development and cope with continuous change. On the other hand, they pose difficulties in scaling up and in integrating with the traditional organisational arrangements that are hierarchical (top-down). In practice, managers face a number of barriers when they attempt to introduce agile methods in organisations that are accustomed to traditional methods. Besides change related challenges, nearly 40 perceived barriers were identified during a workshop that are faced by managers when trying to implement agile methods (Boehm & Turner, 2005). These were first categorised into non-problems, problems that existed with respect to size/scope and significant issues. Based on the latter two categories, the more critical challenges posed were categorised into the following three areas: (1) development process conflicts, (2) business process conflicts, and (3) people conflicts. These can be further subdivided as shown in the table below.

Table 3: Challenges for agile methods in traditional organisations

Development process conflicts	Business process conflicts	People conflicts
Variability Different life cycles Legacy systems Requirements	Human resources Progress measurement Process standard ratings	Management attitudes Logistical issues Handling successful pilots Change management

Development process conflicts include issues related to variability, different life cycles, legacy systems and requirements. Business process conflicts include human resources, progress measurement and process standard ratings. People conflicts include management attitudes, logistical issues, handling successful pilots and change management.

Variability issues arise because the two methodologies tend to lead to “radically different artifacts” that could make integrating difficult thus necessitating some degree of coordination. For example, product functionality or some design details could change. The two also have different life cycles with agile methods focused on delivering immediate functionality whereas traditional methods are concerned with a longer-term optimal development thus requiring adjustments. Legacy systems are difficult to disassemble for accommodating agile replacements. Requirements issues arise because agile methods are usually functional and informal. Human resource issues that tend to arise include timekeeping and position descriptions. For progress measurement, traditional processes based on the earned value tend to be problematic to apply in agile based methods. Process standard ratings are another problematic area, especially for more mature organisations. During migration, processes such as statistical process control stem from a manufacturing paradigm and employees are typically associated with certain roles. Also, collocation is expected for agile teams, systems have to be controlled in a coordinated manner, and they need to be architecturally driven. Compatibility between the organisational structures and processes is another important factor, as are risks throughout the hierarchy. In regard to change management, problems can arise such as obsolescence and inadequacy, dealing with reluctant employees, etc.

Knowing the kind of challenges that could be faced in this scenario is useful because preventive, alleviative and other measures can be taken to facilitate the implementation. This is relevant for UgCD because it shares similarities with agile methods that will be attempted to be introduced in a university in which traditional non-agile methods have

been dominant. Strategies can then be devised to overcome the challenges and ensure the adoption of UgCD is as successful as possible. However, some of the previously identified challenges would be more or less relevant than others in the Saudi cultural and e-learning context so this would need to be taken into account.

2.7 Existing combined agile-plan driven methods

A brief examination is made of some existing methods that fall in between traditional and agile methods including case studies. This is worth making for their possible similarities with UgCD, which also attempts to balance both extremes. One such model is used for the development of Open Source Software (OSS), and another hybrid method is called multiview. Different types of one methodology can also be combined. The combining of more than one agile method for example, was shown to be viable based on empirical evidence in a study by Fitzgerald et al. (2006).

Such traditional-agile hybrid methods are thought to be a good approach to ensure quality development because they combine elements of both as deemed necessary for the software project to succeed. They also mitigate the risks for organisations that have invested heavily in traditional methods but wish to take advantage from applying at least some agile elements (Parsons & Lal, 2006). The reluctance to adopt agile methods in whole is also because agile methods tend to be based more on abstract principles than on concrete guidance (Abrahamsson et al., 2003), and the fear that quality might be compromised due to the lighter approach. Combined methods allow both to be used in a complementary way. In short, the aim is to derive benefits from both approaches to software development and in a way that would not result in losing existing value but would add new value instead.

Hybrid methods take advantage of the fact that the essential goals of software engineering are the same regardless of the nature of the project, i.e. to develop software according to the user requirements at a reasonable cost and within an acceptable time. Thus, the formal and agile methods do not have a conflict of interest; rather, there is a change of emphasis (Parsons & Lal, 2006). As an example, whereas a traditional method might focus on process quality in order to ensure product quality by applying a restrictive process, an agile method might involve a more reflective process to focus directly on the product quality.

2.7.1 The OSS model

The OSS model was seen by Boehm (2002) as lying between agile and what he called

'plan-driven' methods. The table comparing the three with respect to developers, customers, requirements, architecture, refactoring, size and the primary objective is attached in Appendix A. The OSS model actually shares many characteristics in common with agile methods except that the development teams, which are typically larger, are geographically distributed, there is less representation by customers, the software is commonly owned and continuously evolves, the architecture is open, and the primary objective is the challenging problem. UgCD does not need to be like the OSS model because the main development team will be localised at the university rather than dispersed geographically. However, it will share the feature of being a continually evolving product so that it stays in line with the changing user needs of teachers and students. At the same time, it will also resemble plan-driven methods with respect to being designed to meet not only current but also to some extent foreseeable requirements. This could allow some innovative special features of the software to be trialled so that later, they can be either discarded or perfected although this practice will not be treated as compulsory if it is likely to hinder meeting the schedule and cost targets.

2.7.2 Scenario Based design

One approach to software design is to take into account possible scenarios of human-computer interaction. Scenario-based Design (SBD) does this in a way that specifically addresses five technical challenges in software development (Carroll, 2000) that are illustrated in Figure 4. The approaches used in SBD to deal with these challenges indicate that SBD is a combined plan-driven and agile approach because three are plan-driven, while two are agile in nature. The plan-driven components or characteristics are permitting reflection, permitting a variety of interactions, and abstraction and categorisation, and the agile components are flexibility and being work-driven. Scenarios prove useful because they:

(1) Evoke reflection upon the design work content thereby helping developers to coordinate design action with reflection.

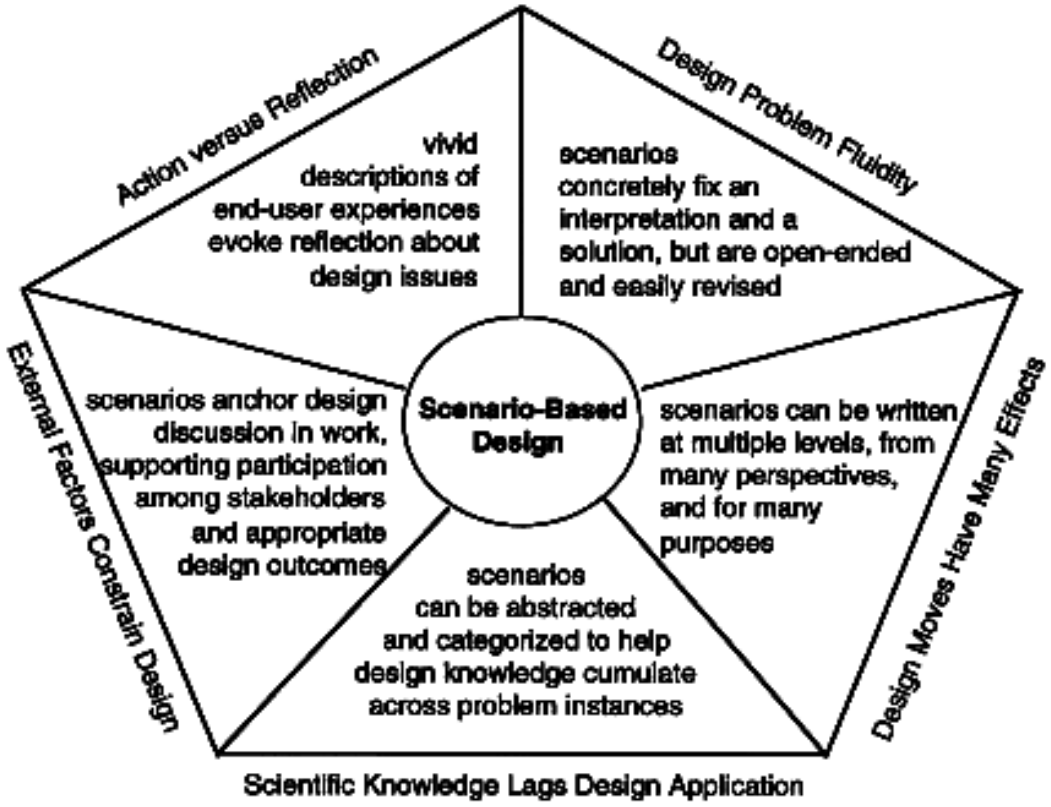
(2) Are immediately concrete and flexible thereby enabling developers to handle fluidity in design situations.

(3) Provide multiple interaction views with a range of detail thereby enabling developers to manage several possible consequences of a design change.

(4) Can be abstracted and categorised thereby enabling designers to recognise, capture and reuse generalisations, and deal with the challenge of technical knowledge lagging behind technical design.

(5) Promote work-oriented communication among stakeholders thereby making design activities more accessible to a range of expertise contributing to design and dealing with the challenge faced by external constraint designers and clients which distract attention from technology users' needs and concerns.

Figure 4: Challenges and approaches in Scenario Based Design



Source: Carroll (2000)

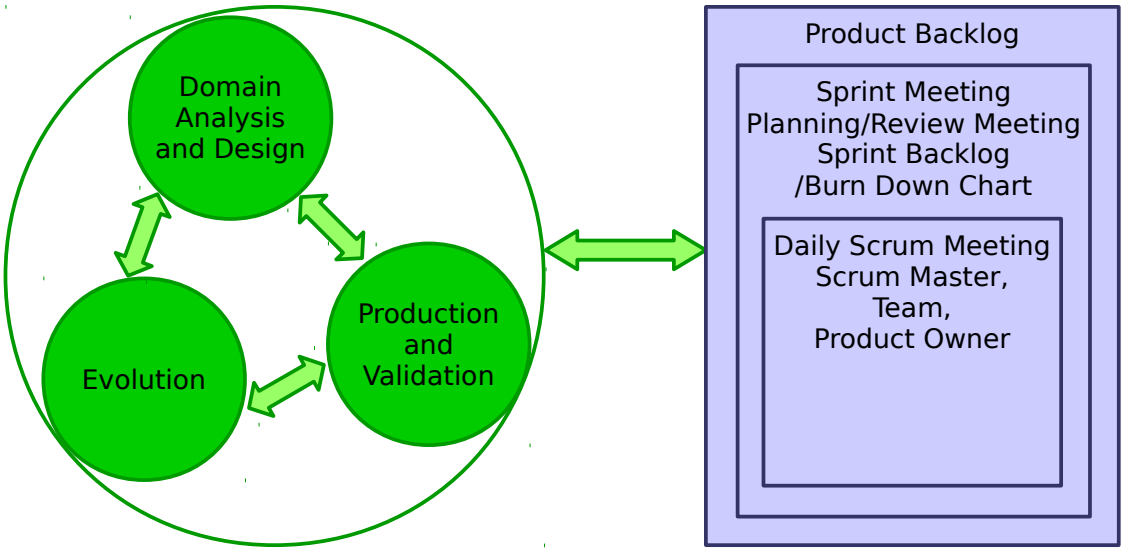
The third identified challenge and potential for SBD is interesting from the point of view of usability because it permits an examination to be made of a range of possible usability consequences in the form of interdependencies and trade-offs. Considering design decisions from different perspectives, including usability, is essential due to the interconnected nature of such decisions.

2.7.3 Recent hybrid development models

Several recent attempts have been made to devise hybrid development models by combining elements of several software development methods characteristic of both plan-driven and agile methodologies. In a study by Silva et al. (2015), 81 other studies were examined that combined CMMI (Capability Maturity Model Integration) with agile software development. Notably, it was found that agile methodologies alone were insufficient for obtaining a certain target rating, and it was necessary to combine them with CMMI. It is not surprising therefore that developers resort to using hybrid models. One early example of a hybrid or balanced plan-driven and agile methods is MBASE (Model-Based System Architecting and Software Engineering) (Boehm et al., 2002).

A more recent attempt at devising a hybrid development method combined three methods, namely RUP with XP and SCRUM (Bashir & Qureshi, 2012). RUP was chosen due to its document driven nature, “high predictability, stability and good quality results for large scale projects” among other reasons, SCRUM due to its ability to manage and track development, and XP due to it having the “best engineering practices”, such as user stories, coding standards, pair programming and metaphors. Their proposed integrated framework (Figure 5) comprises of three major phases and six logical activities. The major phases are: (1) Domain analysis and design, which involves business investigation and design, (2) Production and validation, which involves implementation and testing, and (3) Evolution, which involves deployment and configuration. The framework was designed in order to integrate the strengths of the three separate methodologies “while suppressing their weaknesses”.

Figure 5: A proposed RUP-XP-SCRUM integrated framework



Source: Bashir & Qureshi (2012) (adapted)

Another similar multi-hybrid model also uses RUP and XP, but combined with the AUP (Agile Unified Process) and SPEM (Software Process Engineering Metamodel) used for development for the mobile environment and with the aim of delivering high quality products (Khalid et al., 2014). The Agile Unified Process is a simplified version of the Rational Unified Process.

These frameworks present further evidence of developers acknowledging the strengths and weaknesses of different methods and finding ways to combine them so as to enable them to devise integrated models. Improving usability was not the objective in the above-mentioned study by Bashir & Qureshi (2012), but the need was acknowledged for a methodology that can deal with projects that range in size from small to large scale. The methodology was devised in such a way that more activities typically practiced in RUP alone can easily be inserted as required for larger projects.

2.7.4 Balancing control and flexibility

It can be seen from the above examples that key useful features of traditional methods are that they provide the necessary control and discipline, and key useful features of agile methods are that they provide the necessary flexibility and adaptability. Some degree of control is desirable if not essential, especially in situations of highly specific requirements, high resource constraints and high criticality. On the other hand, some degree of flexibility

is also desirable because it allows for improvisation and the ability to adjust for changes. It is now accepted that sustainable and successful software projects require both discipline and agility (Boehm & Turner, 2003: 23). It would be more accurate however, to state that both qualities are required in varying degrees because software development processes have to be able to adapt to a very wide range of requirements and development environments with culture being another complicating factor. It may seem that the two ideals are incompatible with each other but this is not necessarily the case if some allowance is made for both.

In order to achieve the right balance between control and flexibility, some mechanisms exist such as 'scope boundaries' and 'ongoing feedback'. Harris et al. (2009) refer to these as 'emergent controls' because they allow for a trade-off between control and flexibility by enabling restrictive scope boundaries to be applied and dynamic feedback opportunities to be availed at the same time. Scope boundaries limit the set of feasible solutions without specifying specific outcomes and ongoing feedback provides checkpoints that can be used to validate progress based on the feedbacks (Hevner, 2010: 105). The setting of scope boundaries is a risk management technique which focuses the attention of the development team. The feasible space lies within the boundary and is defined by such practices as having a shared vision, making partial specifications, predefining architectures, determining resource constraints, etc. If this space is restricted too much that it does not allow for any flexibility then it becomes a plan driven approach so the point is to strike the right balance that defines the scope as far as necessary yet allowing for flexibility to shape the final form of the software development phases and product.

As rightly pointed out by Boehm & Turner (2003: 23) in 'Balancing agility and discipline', ascertaining the appropriate amount of rigour or control to be applied is a management issue. Neither the developers nor the end users have a complete grasp of the entire project as they are usually only interested in the phases of relevance, unlike the management, which would normally have a wider perspective. The decision of appropriate amounts would depend on the type of project and therefore also such factors as familiarity and complexity; on imposing factors such as budget, schedule, other resource limitations, degree of criticality, politics, etc., and on the capabilities of the development team according to the personal experiences and degree of team cohesion or collaborative capabilities. As far as the UgCD model is concerned, the type of cultural context is going to be an important factor as well and the type of users are already defined as teachers and

students. Furthermore, the software must be able to enable these two groups of users to work cooperatively and effectively so that the e-learning takes place as effectively and efficiently as possible. Regardless, the point is that for any software project in general, there is a need at the outset to determine the most appropriate balance between control and flexibility.

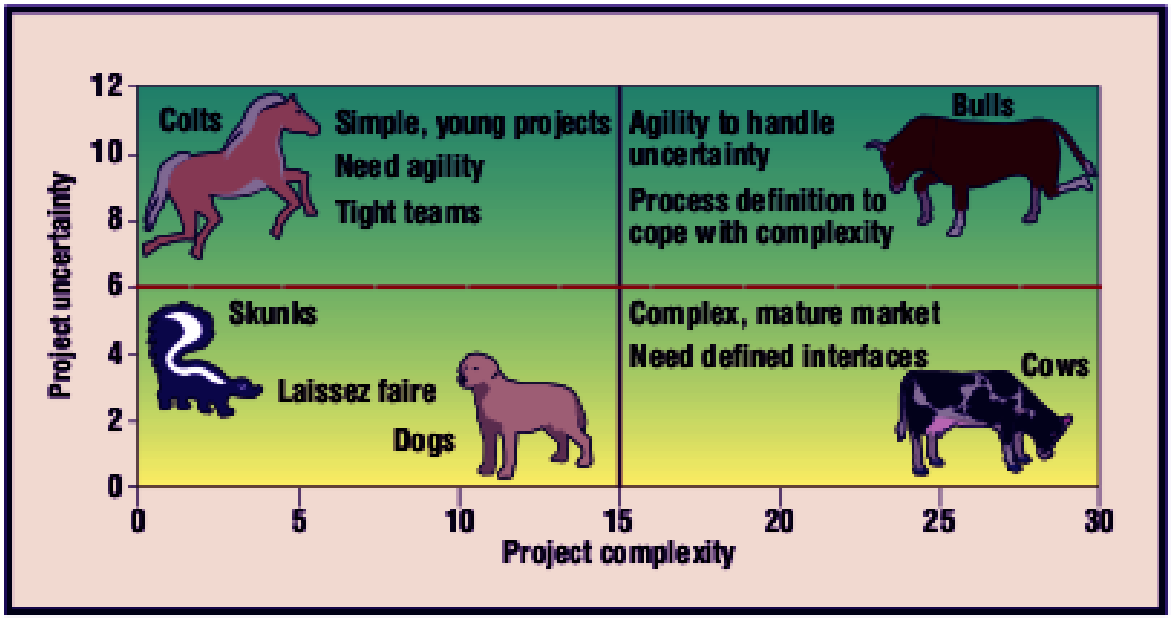
2.8 Achieving the right mix of development practices

In attempting to achieve the right mix of development practices, we need to consider certain features of the project, people and environment. It should be apparent now that projects can differ significantly from one another and certain methods are better suited to certain types of projects and in certain contexts than others. An examination of the methods used in 720 projects showed that the most critical risk driver affecting the success of a software project was an inappropriate development methodology (Tiwana & Keil, 2004), hence the importance of selecting or devising the most appropriate method and not applying the same method for all cases. The issue then is how to determine which traditional or agile based method or elements thereof should be used for a given project, or how to achieve the right balanced approach consisting of the most suitable set of practices. This requires first categorising the project based on its characteristics. Boehm & Turner (2003) identified five such attributes on the basis of which projects can be categorised. These attributes are size, criticality, dynamism, personnel capacity and organisational culture. These can be termed as 'contingencies' because the actual method used can be made to be dependent on these characteristics.

The relative importance of the various characteristics could be an issue in itself and they could also be context dependent. However, a number of them are frequently identified in the literature, not least the five identified above. At one software house, the history of projects were found to have two primary attributes, namely complexity and uncertainty. This led to them developing a four-quadrant matrix that can be used to help in determining suitable additional process practices beyond their core set of practices. They named it 'The Houston Matrix'. The degree of the complexity attribute was ascertained by assessing the following six sub-attributes on a scale of ten: (1) team size, (2) mission criticality, (3) team location, (4) team capacity, (5) domain knowledge gaps and (6) dependencies. The degree of the uncertainty attribute was ascertained likewise by assessing four sub-attributes of (1) market uncertainty, (2) technical uncertainty, (3) project duration, and (4) dependencies

and scope flexibility. Charts showing the varying degrees of these complexity and uncertainty attributes are attached in the Appendix. The company calls these 'project drivers' and identifies these at the outset to guide the teams so they can use the most appropriate process. The 4-quadrant Houston Matrix is shown in Figure 6 below.

Figure 6: Houston Matrix for determining applicable software practices



Source: Little (2005)

This kind of practice in which the developers are able to easily change from one set of development practices to another is indicative of a high degree of adaptability wherein the decision as to which method to adopt is contextual. Some of the factors identified by Boehm & Turner (2003) are combined along with others to measure the degree of 'complexity' so they are not treated separately. This scheme bears some resemblance to the Boston Matrix, which is a tool used in product portfolio management (McDonald & Wilson, 2011: 172). Notably, despite adding to the five attributes of Boehm & Turner, organisational culture is not included. This is presumably because their software development processes have never needed to be adjusted to handle different cultures. An attribute of 'team location' is included but it takes into account the degree of dispersion, not cultural differences.

For a more universal scheme, it would therefore be necessary to include the fifth attribute

of Bohem & Turner, i.e. of organisational culture, and perhaps also the wider national culture. This suggests that not only is it necessary to recognise the characteristics of the project but that it is also necessary to consider the people involved, both with respect to the development of the project and its end users, and the wider environmental factors as well.

2.8.1 Consideration of project, people and pressures

In preparing a suitable software development methodology, the context-adaptive UgCD recognises the importance of not only the technical parameters of the project but also the social aspects relating to the people involved and other environmental or external aspects. This equal importance was also recognised in a survey in Malaysia, i.e. that both technical and social aspects must be considered for ensuring benefits from agile methods (Asnawi et al., 2010). These components for consideration have been termed as Project, People and Pressures respectively and they are referred to collectively as the '3Ps'. That is, the form of UgCD to be applied will be shaped by the project characteristics descriptive of the project itself; by the people involved in terms of their degree of empowerment, communicative ability, experience, user needs, entrepreneurial skill, and so on, as well as by the pressures of cost and time constraints, organisational culture, legal restrictions, political issues, etc. Together, all these considerations should be able to account for all the important attributes that impinge on the decision as to which development method and processes to use or devise for a particular software development project.

Chapter Three

Usage-Centred Software Design and Development

Chapter 3: Usage-centred software design and development

3.1 Introduction to Usage-centred Design

The concept of 'Usage-Centred Design' was coined by Constantine & Lockwood. It has similarities with Activity-Centred Design (ACD) and Joint Application Design (JAD). The issue is that people are concentrating more on integrating agile methods with user-centred design nowadays but not so much on ensuring the software satisfies its primary purpose. In the case of e-learning software for example, this primary focus would be on meeting the actual learning needs of the users (i.e. teachers and students) over other concerns relating to user preferences. Usage-centred design thus focuses on the tasks to be accomplished (which in the aforementioned case mainly relate to learning) instead of on the users per se. User interface issues are also given consideration because they facilitate usage tasks.

Usage-centred designs arise from the need to deal with pragmatic concerns effectively and efficiently. They involve a systematic process geared to completely and directly supporting all the tasks that users are required to accomplish. The objective of the usage-centred approach is to attempt to simultaneously facilitate efficiency, reproducibility and creativity from the process of design (Constantine, 2003). Three types of (interrelated) models are developed during this approach, to identify the roles, tasks and content, as follows (Constantine & Lockwood, 2001):

- Roles – to capture the characteristics of the roles of users in the system
- Tasks – to represent the work structure that users would need to accomplish
- Content – to represent the interface contents and organisation for supporting the tasks

Constantine (2003) thus describes it as a 'model-driven approach' because the design derives from a set of core models directly and systematically. Although models also exist in more traditional methodologies, these models are designed “to provide the most conceptual and creative leverage for the least amount of effort on the part of analysts and designers”. Unlike in the Rational Unified Process, they are designed to express tasks in the simplest ways possible. As such, the emphasis on modelling differentiates the usage-centred approach from pure agile methods although Constantine himself describes it as an adjunct to agile processes as it helps develop highly usable user interfaces. After all, it also

relies heavily on communication and close collaboration with the end users except that it is ensures the users are genuine and the experts are true experts in their domain.

The idea behind UgCD is that instead of building software around users, the software should be built around *uses* so that it can better target what users are trying to achieve. According to Norman (2005), who incidentally coined the phrase 'user-centred design', in order to create software that is highly usable, “the most important issue is neither the user nor the user interface, but usage”. As such, usage-centred design often leads to innovative solutions due to the attention given to improving usability. Examples of some techniques that can be used to make interaction with the interface more instructional (for improving its usability) are shown in Appendix A.

3.1.1 Examples of the application of usage-centred designs

Usage-centred designs have been applied in the design of several projects. For example, it has been used in the development of a software for a PDA for gathering field data (Moe et al., 2004). Constantine & Lockwood (2002b) mention a wide range of applications from banking to industrial automation systems and in the development of a browser-based classroom management system, which needed to be developed in a short time and in which the project had to face further risk factors.

3.2 Problems with agile methods and user-centred design

Agile methods are averse to detailed requirements gathering, which is not the case with UgCD. The limited requirements gathering in SCRUM for example, is based on an incremental unfolding so the task is broken down into manageable portions. User stories are used to gather the changing requirements. However, having an on-site customer could be too much to expect from teachers/students. Besides, knowing how to handle requirements in agile methods remains an issue (Kavitha & Thomas, 2011). For a learning environment. Ovesen et al. (2011) identified the following problems with particular agile methods:

- Scrum is not so informative on how to specifically perform a certain task
- XP is stronger on concrete guidance and procedures
- Feature-driven development provides no concrete methods for specific steps

- Pragmatic programming provides advices and takes the guidance to “a more principle level”

Although the findings were in relation to the development of a learning environment, they could perhaps be a common experience during the development of software in general. Another particular problem for all agile methods is that of non-scalability. That is, agile methods are not easily scalable to larger projects, which is not a problem for UgCD. A study by Schneider & Johnston (2005) for example, showed that XP is not suitable for large-scale system development of academic software in tertiary education. Agile methods are also not effective for GUI intensive applications with the exception to some extent of DSDM and FDD (Constantine, 2002).

User-centred designs (UrCD) rely heavily on user studies and therefore tend to confuse and focus unduly on what the users want or prefer rather than on what they actually need more importantly. They also involve rapid iterative prototyping, which could perhaps be better avoided by having a good design in the first place. In the same way, the need for extensive usability testing can also be avoided.

Another fundamental problem with user or 'human-centred design' is that the developed software tends to suit only certain people, particularly those involved in its development. This makes it less likely to be suitable for users generally. Moreover, it detracts from supporting the more important activities and can lead to “a lack of cohesion and added complexity in the design” (Norman, 2005). This is especially evident in software that involves numerous, often overlapping tasks, and sequences of operations. This does not mean that users should be ignored but that for the sake of coherence and general applicability, someone responsible in the design team should have the ultimate say and make the final decision taking different views into account.

Such a design approach is followed, for example, by Apple Computer, which is epitomised in the following words of the former CEO Steve Jobs and is reflected across all its exemplary products: “*It’s really hard to design products by focus groups. A lot of times, people don’t know what they want until you show it to them.*” (Steve Jobs). Although this was said in reference to computer hardware, the concept could also be applicable to the HCI and computer software generally.

3.3 Differences between user-centred and usage-centred design

In comparison with user-centred design, usage-centred design, as the name suggests, is focused on usage rather than on users per se. Users are still recognised but their involvement is selective and through exploratory modelling, model validation and structured whereas in user-centred design, it is more substantial and through user studies, participatory design, user feedback and user testing. User-centred design also involves descriptions of users and identifying user characteristics whereas usage-centred design involves making models of user relationships with the system.

Being a model-driven approach, usage-centred design is also more systematic whereas the processes in user-centred design are often unspecified, informal and varied. This makes usage-centred design better grounded in software engineering, especially due to its object-oriented approach and its reliance on extensions and refinements whereas user-centred design is more firmly grounded in human-computer interaction and other human factors.

The method of design is also fundamentally different. User-centred design involves making successive approximations following cycles of trial and error and extensive user testing. In contrast, usage-centred design aims to produce the right design from the outset although it may later be improved and refined. The above-mentioned differences are summarised in the table below.

Table 4: Key differences between user-centred and usage-centred design

	User-centred (UrCD)	Usage-centred (UgCD)
Focus	Users (people)	Usages (activities)
Objectives	Improve user experience/satisfaction	Improve tools to support task accomplishment
Driven by	User input	Models
User involvement	Substantial	Selective
Identification	Of user descriptions and user characteristics	Of user relationships with the system
Design models	Realistic or representational	Abstract
Method of design	By iterative prototyping	Through modelling
Processes	Usually unspecified or informal and varied	Fully specified and systematic
Design progression	Evolution by trial and error	Derivation by engineering
Coherence	Tends to be lower	Tends to be greater
Popularity	Dominant	Less popular

Sources: Constantine & Lockwood (2001), Constantine et al. (2003) & Norman (2005)

3.4 Human-centred designs

The existence of UrCD, also known as UCSD (User-Centred Systems Design), is indicative of the trend of what can be termed as 'human-centred software engineering'. UrCD promotes the view that users needs and capabilities are critical, and users are regarded as being at the centre of the development process. This approach is particularly suited for any software that involves HCI (Human-Computer Interaction). Initially, UrCD emerged to promote the practice of merely studying, observing and measuring users for the purpose of gathering requirements rather than involving them in the development process (Bernhaupt, 2010: 47). By doing so, developers would then obtain sufficient feedback to help them in developing the software according to user needs.

Under UgCD, the involvement of users is taken much further to the extent that they actively collaborate in the development process. As such, UgCD bears greater similarity to to a more radical Scandinavian variety of UrCD known as Participatory Design (PD). In PD, users are treated as equal partners in the software development process. PD is also referred to as Cooperative Design or Co-Design. PD invovles focusing on primary work processes and identifying technologies to better support or enhance work activities.

Although involving users during software design has become more widely accepted nowadays, as Marti & Bannon (2009) pointed out, this practice can also present certain problems. Their study showed difficulties especially in working with children and those with differing mental abilities, but they acknowledged the importance of a UrCD approach as long as it fits the context for which it is applied. The use of personas in UrCD tries to overcome these limitations, but personas do not tend to be determined using real data (Masanari, 2010), and they are by nature abstract and can be impersonal, misleading and distracting (Mathews et al., 2012). UgCD overcomes these limitations by focusing not on users per se, but rather on usage, i.e. the uses for which the users will be using the software to accomplish certain tasks. As pointed out by Kelly (2014), user-centredness obscures the concept of use in design, as tasks, duties and functions are more important to understanding relationships that are of relevance to design.

Since software is created by humans for humans, a human-centric approach to software design is essential as opposed to a purely artifact-centric one so that human intelligence is amplified (Murphy, 2010). Furthermore, usability concerns especially, are best addressed not by adding usability enhancing activites to and existing process, but to make it a natural part of it (Goransson et al., 2003). UrCD does involve users (substantially) and is driven by user input, but its objective is to improve the user experience or satisfaction rather than supporting the accomplishment of essential tasks. Constantine described UrCD as a “loose collection of human-factors techniques united under a philosophy of understanding users and involving them in design… although helpful, none of these techniques can replace good design. User studies can easily confuse what users want with what they truly need.” (Constantine & Lockwood, 2001: 43).

While agreeing with Constantine's aforementioned critique of UrCD, Goransson et al. (2003) regard UgCD as moving away from UrCD by reducing their involvement to that of being informants instead of co-designers. However, it was established above that UgCD is more in line with PD as it involves users in a more participatory or collaborative way than is typical of traditional user-centred designs. Moreover, usability engineering is embedded in UgCD rather than being an addition, which is what they advocated. Also, they rightly pointed out that involving users can be complicated and time-consuming, but the time factor may be a necessary compromise if usability is to be ensured, as involving users is essential. The real question is how effectively users are involved. UgCD involves them more selectively while focused on facilitating task completion through enhancing usability.

3.5 Advantages of usage-centred design

Patton (2003) showed how adding the usage-centred design approach in an agile software development environment can make the process run smoother. Figure 7 below shows an XP based design with (bottom) and without (top) usage-centred design components. At each of the marked points, usage-centred design can provide a potential solution to potential difficulties.

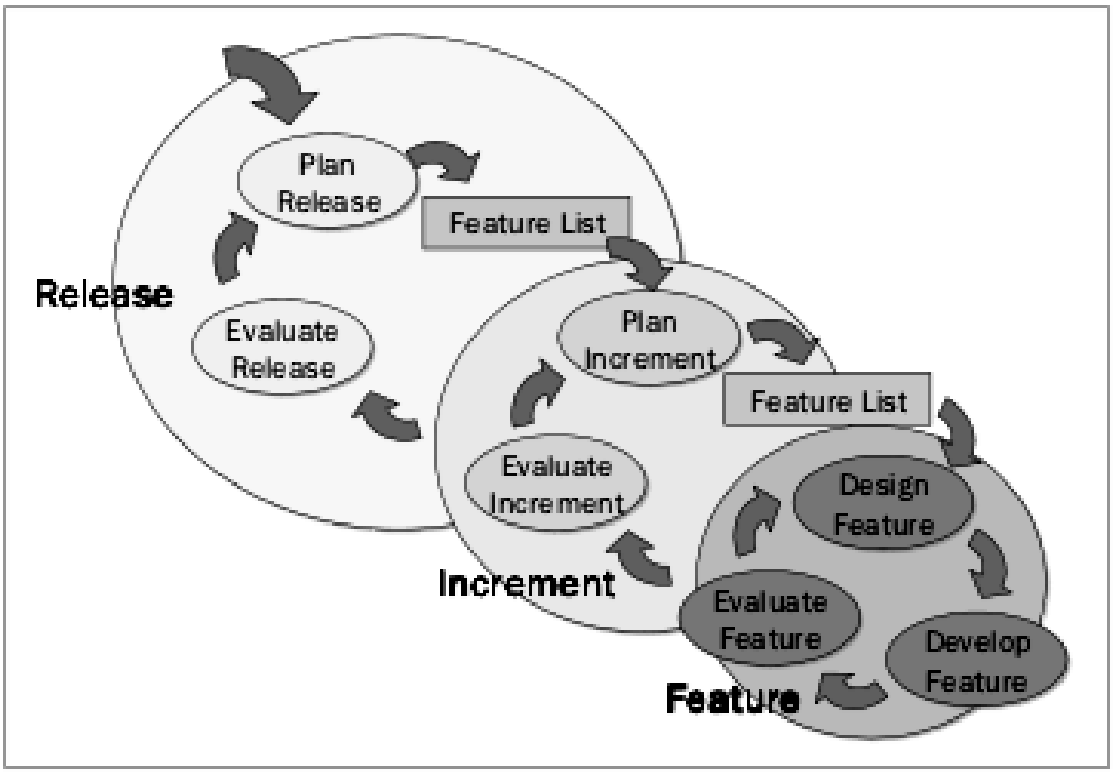
In this model, incremental planning is done collaboratively involving both developers and customers among other stakeholders. The plan itself includes mention of features that can be reasonably expected during the increment and the process involves some requirements gathering or a simple design to arrive at the features. In XP, user stories help to determine these features, which are then labelled and prioritised. During the designing, it is common for a customer to be on site with the development team, and there is an emphasis on quality throughout.

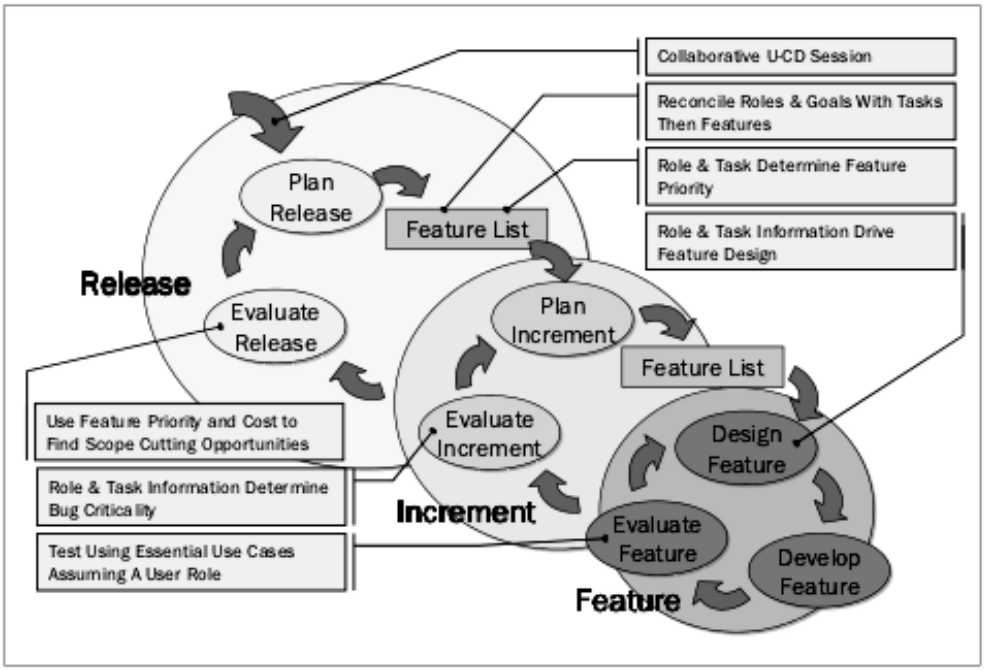
The element of UgCD in this design and development process helps particularly when difficult design decisions and trade-offs have to be made. In summary, UgCD principles can be usefully applied in an overall agile framework in the following ways:

- To facilitate collaborative sessions so as to make the best use of the time of the important people involved by using card based techniques.
- To aid in determining the priority of features based on user roles and tasks
- To ensure that all tasks are handled by important features by reconciling the roles, tasks and goals
- To help in determining how each feature will look and behave in the software using essential use cases
- To test the software by assuming user roles and writing test cases

Usage-centred design has been shown to be particularly effective for complex situations in which user performance is critical (Constantine, 2003).

Figure 7: XP based design with (top)/without UgCD components





Source: Patton (2003)

Chapter Four

Usability in Software Design

Chapter 4: Usability in software design

4.1 Software quality

4.1.1 Indicators of software quality

The International Organization for Standardization and The International Electro-Technical Commission (ISO/IEC 9126-1) identify the following six categories of software quality attributes:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

These attributes are interrelated, so for example, reliability and efficiency also impinge upon usability, as does learnability, which is not identified above. Also, some are greater concerns for users, as is usability, whereas maintainability, for example, is a greater concern for designers or administrators. According to this standard, usability is defined as "the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions". (UsabilityNet, 2006). In this definition, learnability is implied though not explicitly identified on par with usability; rather it is one of the sub-attributes, the others being understandability, operability and attractiveness. The quality of understandability was examined, for example, in a study that evaluated a government website (Zhou, 2009). It was noted that usability engineering and information architecture should be given due attention when designing a website based on usage-centred design.

4.1.2 Potential of UgCD in improving software quality

UgCD has been shown to be advantageous in terms of improving the quality of usability of software. This is a quality that is typically deficient when implementing agile development

methods, as recognised by Constantine among others, although this does not necessarily need to be the case (Fuller, 2004). It is therefore often necessary to incorporate elements of Usability Engineering (UE) and/or Interaction Design (IxD) into agile software development methods (Barksdale & McCrickard, 2012: 53). The incorporation of usability into an agile method is referred to as 'agile usability'. Constantine's UgCD approach does this through card-based modelling and involving decision making whilst remaining a lightweight, iterative and incremental process. Others have also approached this integration challenge in different ways.

UgCD also has the potential to improve other qualities of the software being developed besides usability. For example, it was mentioned earlier how incorporating the UgCD approach in an agile software development environment can make the process run more smoothly. In this study by Patton (2003), a comparison was made between an XP based design with and without UgCD components. In the former, efficiency in developing software was improved as the collaborative sessions that utilised card based techniques enabled time to be used more efficiently, and the defining of user roles and tasks enabled software features to be easily determined and prioritised. In short, based on an agile framework, UgCD can provide 'a powerful punch' through its iterative process to deliver software that is of high quality and which meets user expectations (Patton, 2002).

Another major advantage of a UgCD approach for software development is that of scalability, as it is also for traditional plan-driven methods but not for agile methods generally. In fact, UgCD was specially designed to be scalable, and the reasons for it being capable of being so are several (Constantine & Windl, 2003). In short, there has been a complete working out and refinement of both highly simplified forms in which roles and tasks are merely identified, and highly structured models in which roles and task cases can accommodate large and complex applications. This enables simple arrangements to be easily adapted to become more detailed models as and when needed.

Another quality that is an inherent capability of UgCD is that of functionality, as it is defined by use cases (Gulliksen & Boivie, 2001: 10). These use cases attempt to capture what is needed to be known about the context of the software development project and specifically user needs. As such, determining functionality is the starting point in UgCD. Defining user roles is also a central feature of UgCD, as mentioned earlier. Identifying these roles, as well as the relationships between them, helps to guide the decisions about the functionality that will be incorporated into the design (Ferreira et al., 2005: 7).

The table below lists the key studies that have either demonstrated empirical evidence of improvement in a certain quality using UgCD, or provided an ample theoretical basis linking the quality to UgCD. There is a need to clarify the distinction between satisfying certain users and of fulfilling more generally applicable usage requirements, and in emphasising the importance of the latter.

Table 5: Key studies on different quality aspects of UgCD

Software quality attribute	**Previous empirical studies that have showed how the quality is improved by UgCD**	**Previous theoretical research that has linked the quality with UgCD**
Functionality	-	Ferreira et al. (2005)
Reliability	-	-
Usability	Dabbagh (2012)	Constantine & Lockwood (2002b); Constantine (2002); Barksdale & McCrickard (2012)
Efficiency	Patton (2003)	-
Maintainability	-	-
Portability	-	-
Scalability	-	Constantine & Windl (2003)

4.2 Introduction to usability

According to the IEEE Standard 1061 (1992), usability usually refers to how easy it is for a user to learn to operate, to prepare inputs for and to interpret the outputs of a system. In simple terms and in the context of software engineering, this means the ease in learning to use, actually using and comprehending the use of a piece of software. It therefore involves learnability, but it is also concerned with efficiency and effectiveness, i.e. of using the software, and overall satisfaction or rather intuitiveness.

Usability thus comprises of the quality attributes of learnability, effectiveness, and satisfaction. Bevan, N. & Azuma, M. (1997) identified the sub-attributes as performance, satisfaction, and learnability instead, and pointed out that usability refers to such attributes, i.e. either individually, or to all of them together. As such, there has been no consistent understanding of what constitutes usability within the software development industry in practice. These differences in perception are reflected in the two further different standards below, which are in addition to the aforementioned one, and were highlighted by Seffah &

Metzker (2004):

> "The capability of the software product to be understood, learned, used, and [made] attractive to the user, when used under specified conditions." (ISO 9126)
>
> "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." (ISO/DIS 9241-11)

The first definition emphasises learnability and satisfaction, but not effectiveness, and the second emphasises effectiveness and satisfaction, but not learnability. In contrast, the previous definition is focused on learnability aspects, implies the need for effectiveness, but does not specify satisfaction. Newer definitions give greater emphasis to the satisfaction aspect, which is in line with the concept of 'user satisfaction'. For example, Rubin & Chisnell (2008: 4) describe usability as an invisible quality, whose absence would be noticed and would cause dissatisfaction. In fact, they regard usability as a sub-attribute of usefulness, which incorporates attributes of learnability, efficiency, and satisfaction among others.

As far as this study is concerned, the researcher will consider usability to incorporate all three sub-attributes of learnability, efficiency and satisfaction besides reliability and rememberability. The key definitions and studies supporting each attribute in order to justify its selection are given in Table 18 in section 6.8.1 Selected usability aspects restated. This table covers all of the aforementioned attributes as found in several supporting studies.

4.3 The importance of usability

Usability is a very important issue in HCI (Human Computer Interaction). As Larman (2002) noted, usability is a very important feature for a software to be successful, yet often it is not given sufficient rigorous attention. Usability is a very important consideration, especially for e-learning software as it helps to develop better systems with improved didactical and pedagogical approaches (Novak et al., 2010). It affects for example, the quality of our experience in operating computers and other computer based devices including smartphones, in using software and in browsing the Internet. Among these, there is greater variety in web-interfaces which people interact with to browse websites. Constantine (2011) observed that many websites especially, sacrifice usability for a quality

of “visual razzle-dazzle”. This is not unexpected though given that websites are designed by far more numerous organisations and individuals and whilst there are some common layouts, formats and elements, there are always bound to be some websites that seek to differentiate themselves. This is not necessarily a bad thing because the context, i.e. content, purpose of site and type of visitors are also heterogenous and could have specific needs.

By improving usability, software becomes more usable. For example, it could result in less time spent in learning how to use the software, which improves learnability. For businesses, improved usability could lead to reduced training and cost savings. Usability could therefore lead to improvements in productivity, quality of work and greater user satisfaction. In the case of websites, good usability is less likely to lead to the users switching to other alternative websites. Furthermore, it is important to accommodate for usability and detect potential usability issues as early as possible. This can prevent, for example, costly late-cycle changes (Memmel et al., 2007).

4.4 Usability and other software quality characteristics in agile methods

Usability, or rather the lack of it, is an issue when pure agile methods are used for software development because, despite their success otherwise, they lack the capability of ensuring software usability (Kane, 2003). This lacking was also noted by Larry Constantine (2002), who was more vociferous in stating that agile methodologies ignore usability for end-users, and who popularised UgCD (Usage-Centred Design) instead. In fact, the agile manifesto, which popularised agile methods, did not even incorporate usability engineering, so unless additional design elements are included, software developed using agile methods alone are not usually renowned for their usability. Consequently, various attempts have been made more recently to integrate elements to ensure usability into agile methods, termed as UrCD (User-Centred Design).

However, the same is the case for techniques that purport to be based on UrCD. The reason for this is that these techniques were developed independently from the traditional techniques already established in the software engineering community, which do incorporate usability concerns more stringently (Seffah & Metzker, 2004). Nonetheless, if agile methodologies are to remain in vogue, improved methods for this integration must be sought with a focus on developing highly usable software, or else, as this study advocates, a UgCD design approach would need to be implemented instead, under which usage

concerns are paramount.

4.4.1 XP and Scrum

XP focuses on customer satisfaction through rapidly creating high-value software using development techniques based on skill, sustainability, and the flexibility to adapt to change (Larman, 2003: 139). XP is designed to improve software projects in five main respects: communication, simplicity, feedback, respect and courage (Kavitha & Thomas, 2011). Besides prioritising user satisfaction, the approach to gathering requirements and designing software also attempts to satisfy functionality requirements.

Scrum is also guided by five values. Besides respect and courage as in XP, Scrum values commitment, focus and openness. The Scrum approach involves managing feature requests, and handling the features to be planned and included in the final software. It does not appear that there are any attempts to specifically satisfy any of the identified software qualities.

The inability of XP and Scrum to give greater attention to improving the quality of the software such as usability probably has its roots in one of the four principles enshrined in the Agile Manifesto, namely 'Individuals and interactions over processes and tools'. People are considered more important than the software itself, or to put it another way, the user is given precedence over usage. On the other hand, a survey by VersionOne (2008) on the benefits of agile methodologies revealed more than two-thirds (68%) of respondents did agree that agile practices actually lead to enhanced software quality. Another survey on Agile Adoption Rate, found this proportion to be even higher at 77% when comparing with traditional methods (Ambler, 2008). However, in the first survey, benefits such as improved project visibility, team morale and productivity were perceived as having improved more.

It is thus not uncommon for agile methods to be combined with UrCD for ensuring user centricity and usability. This observation alone indicates that without the incorporation of additional methodological elements, agile methods alone are inadequate for ensuring usability of the software produced. This has been noted, for example, by Singh (2008) in examining Scrum, and by Jokela & Abrahamsson (2004) in examining XP. From the perspective of proponents of UrCD, agile methods do though have the potential to provide a solid basis for a user-centric attitude due to their focus on customer collaboration,

communication, customer needs, etc. (Blomkvist, 2005). This is despite there also being some aspects that are not conducive to a user-centred attitude, such as short iterations, lack of distinction between customers and users, an over focus on programming, and unsatisfactory techniques for modelling tasks and users (Blomkvist, 2005). Also, if progress in these integration efforts are to be sustained, then the approach to adopting agile methods must remain wide such that a system development perspective is taken, and not narrowed by seeing it merely as a programming methodology (Nodder & Nielsen, 2009: 7).

4.4.2 eXtreme Scenario-based Design

One attempt to modify an agile method to deal with usability concerns is called eXtreme Scenario-based Design (XSBD) (Lee et al., 2011). This agile usability approach was developed to facilitate enterprise level organisations working with distributed development teams accustomed to agile methods but concerned about usability. The combined approach integrated both agile and usability elements by streamlining usability and development practices, and adopted clearly defined communication and information sharing practices. XSBD makes use of a Central Design Record (CDR) for representing main user experience, and involves usability testing to verify the production of a usable system meeting high level design goals, identify any problems, and to uncover any new requirements. Both development and usability tracks are treated separately but are synchronised with the usability engineer being one step ahead of developers. The study found that many challenges were experienced with communication, collaboration and information sharing, and that XSBD is hampered by the distribution of the development team. However, the CDR was found to be useful for focusing on the most critical areas to ensure high level goals are met. The late uncovering of new requirements especially, suggests the need for more initial planning and early gathering of requirements. This would necessitate combining usability approaches not with an agile method but with an approach more in harmony with usability goals; that values processes or the tasks to be performed more highly than social concerns, as is possible with UgCD.

4.5 Usability in usage-centred design

A software is merely a tool to achieve something and usability is one of the important qualities of the software. The tool can be described as 'usable' if it makes the task for which the tool has been prepared easy, simple, flexible, quick, etc. These aspects are not focused

upon in user-centred design. However, in usage-centred design, it is considered important while designing the software to take into account what the actual purpose of the software is so as to ensure the software is highly usable first and foremost.

There is a dearth of literature available on usability studies grounded in a usage-design context. However, one recent study was found that used a usage-centred design method for developing an e-learning software focused on meeting user needs in which usability testing was undertaken (Dabbagh, 2012). In this study, there were two rounds of testing and each had two phases. The first round involved immersion and proxy participant feedback and the second round involved end-user participant feedback. The latter involved nine participants being given six task-based scenarios to accomplish and their thinking processes were captured as they were made to think aloud. Improvements were then made based on the comments received. The exercise helped to deal with design inconsistencies and other usability issues that affected user performance and user satisfaction. The layout, navigation and functionality elements of the software were praised.

4.6 Historical overview of user/usage-centred development practices

Focusing on uses and usability are not new concepts however, and they are best dealt with at an early stage during requirements gathering. Early attempts to determine software requirements based on the use of a software were made by the likes of Rebecca Wirfs-Brock and Meilir Page-Jones in which scenarios and stories were used (Kulak & Guiney, 2012). Ivar Jacobson then introduced the concept of 'use cases' in 'Object-Oriented Software Engineering' in his book by the same name, which later became part of the Rational Unified Process (RUP). Provision for use cases through diagrams is now included in the standard Unified Modeling Language (UML), which provides the nomenclature for software specifications and diagrams. These use cases are “simplified and generalized down to the essential core of usage” (Constantine, 2000: 5) and usage-centred design makes use of essential use cases along with user roles.

Besides the focus on uses, another characteristic of usage-centred design is the use of some modelling during the design phase. The modelling is not extensive as in traditional plan-driven methods; rather it is applied to essential use cases only. This practice has its roots in Steve McMenamin and John Palmer's concept of 'Essential Modelling'. In this, the focus is on *what* the software is designed to do as an aid to conceptualising the processes (Brown, 2008). The purpose, as in usage-centred design, is to ensure high usability of the software.

4.7 Software design for usability

In software design, usability can be considered either as an afterthought or during the design phase. The latter would probably better ensure that the software quality requirements are met. Unfortunately, usability along with ensuring user-friendliness is often a neglected area, especially in e-learning software design and implementation (Kruse, 2002). Even if the quality of a software is otherwise high, it does necessarily mean that its usability is also high so usability aspects need to be specifically considered and incorporated. Applying usability techniques in software design can lead to enhanced user efficiency and satisfaction, which in turn could lead to greater productivity (Ferre & Juristo, 2001). In fact, usability is often critical for user system acceptance.

The classical attributes of usability are: (1) learnability, (2) efficiency of use, (3) reliability in use, and (4) satisfaction. Such attributes are particularly suited to the context of user-interface design. For software in general, many more attributes of usability can be added. Juristo et al. (2003), for example, mention consistency, natural mapping, accessibility, error management, etc. and Gonzalez et al (2010) include effectiveness and engaging, and to this list can also be added operability, rememberability and motivating, which have been mentioned by others. These various usability aspects are summarised in the table below.

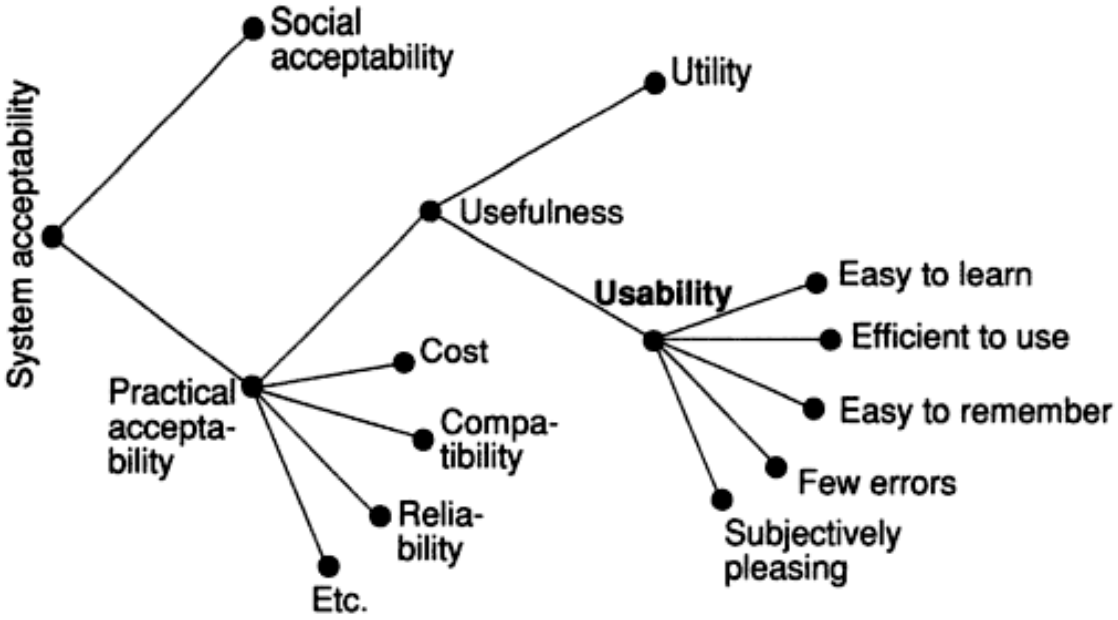
In web based software, usability is probably an even greater issue because it is a more competitive environment than for non-web based software, as users can easily switch to alternative websites (Lilburne et al., 2004) and do so immediately.

Table 6: Usability components

Established attributes		
Classical attributes	**Nielsen (1994)**	**ISO9241**
• Learnability • Efficiency • Reliability • Satisfaction	• Easy to learn • Efficient to use • Easy to remember • Few errors • Subjectively pleasing	• Effectiveness • Efficiency • Satisfaction
Additional attributes		
Juristo et al. (2003)	**Gonzalez et al. (2010)**	**Other**
• Consistency • Natural mapping • Accessibility • Error management	• Effectiveness • Engaging	• Operability • Rememberability • Motivating

The second column in the above table lists Nielsen's (1994) five dimensional breakdown of usability that are illustrated in Figure 8 below. These are the expected software qualities, whereas the actual dimensions related to each of them were specified as learnability, efficiency, memorability, (few) errors, and satisfaction. Learnability refers to the capability of the system to be easy to learn such that a user can get work done quickly by using it; efficiency refers to enabling high productivity to be gained from using it efficiently, and memorability refers to enabling the user to easily remember the system so that there is no need to learn everything when using it again after a lapse of time. The other two requirements require the system to have a low error rate and for it to be pleasant to use.

Figure 8: Nielsen's five dimensional breakdown of usability



Source: Nielsen (1994: 25)

4.7.1 Usability aspects for e-learning software

For e-learning software, the learnability aspect is one of the most important measures of usability (Kobbenes & Folkman, 2003), In addition, other important aspects of usability for e-learning software are effectiveness, efficiency and satisfaction. For e-learning software under UgCD, Constantine (2011) highlighted the following usability aspects for ensuring a highly usable e-learning system: learnability, rememberability, efficiency in use, reliability in use, and user satisfaction. Others have also emphasised the motivational aspect of usability by incorporating elements of feedback, comprehensiveness, challenges, and curiosity (Shilwant & Haggarty, 2005). Costabile et al. (2005) elaborated on these elements to state that an effective e-learning system should provide feedback and be interactive, have specific goals, be able to motivate users, provide suitable tools, help to avoid any interruptions or distraction to learning, and be able to provide a continuous feeling of challenge. As overall objectives, Venkatesh et al. (2003) mentioned that an e-learning system is usable only if it is easy to use and is useful for learners in helping them to accomplish their learning tasks. If students benefit through using an educational software in terms of improvements in their learning, then the software can be described as having a

high utility. As Nielsen (1994) pointed out though, utility is distinct from usability in that utility concerns whether a system's functionality can do what is required whereas usability is concerned with how well users can use the functionality (p. 25).

4.8 Usability testing and measurement

4.8.1 Usability testing practice

As pointed out by Constantine & Lockwood (2002), ultimately, the success of any design is determined by its acceptance by the user, and a key factor that makes a software acceptable is dependent upon its usability. According to usability researcher Jakob Nielsen, 85% of usability errors can be identified within the first five usability tests (Stanley & Kurtz, 2011), hence the importance of at least five usability tests during software development. In practice, ensuring usability is not straightforward due to difficulties in the testing and measuring of usability.

A possible explanation for why usability issues still prevail to a large extent is that usability testing is considered to be a tedious, least rewarding and more expensive of tests to implement (Mueller, 2009). Ensuring usability however, requires special training and skills, and close coordination between the designers and programmers. Evidence suggests that although many software development organisations do engage in usability evaluation, its resource demands and developer mindsets are two significant obstacles (Bak et al., 2008). Moreover, usability is a quality factor that does not lend itself to easy testing as do time and cost based factors, such as performance metrics. In fact, the various factors are inter-related because performance, stability, productivity, etc. also determine a software's usability. It is important therefore that a measure of usability is applied.

4.8.2 Usability testing methods and instruments

Simple methods for checking usability are Cognitive Walkthrough (CW) and 'expert evaluation' while agile methods give importance to user involvement (Wurdel, 2011: 34). Other widely used methods are Think Aloud (TA), which is usually lab-based, and Heuristic Evaluation (HE), which is a cheap and simple method but one that is only normally suitable for low-priority problems of usability (Hwang & Salvendy, 2010). In comparison, CW is a theoretical method, which is designed to discover mismatches between conceptualisations of a task between designers and users. However, it requires an extensive knowledge of technical details and cognitive psychology to apply. One usability

assessment technique is known as SALUTA (Scenario-based Architecture level UsabiliTy Assessment). It is used to assess whether a designed software architecture meets usability requirements (Folmer et al., 2004).

Several questionnaire based instruments are also available for measuring usability. Examples of these are the Computer Usability Satisfaction Questionnaire (CUSQ), Questionnaire for User Interaction Satisfaction (QUIS), Software Usability Measurement Inventory (SUMI), and the System Usability Scale (SUS). These are post-study questionnaires as opposed to other questionnaires that are used, for instance, for assessing perceived usability during use, such as the Website Analysis and Measurement Inventory (WAMMI).

For measuring usability, it is not necessary to use complex or expensive procedures though. Mueller (2009) for example, demonstrated that simple and traditional techniques can also be applied using scenario based testing as a means to measure a user's productivity and learnability. It is pertinent to note that tasks were first defined to produce test cases for measuring productivity and learnability. Usage-centred design also sees the ability to carry out tasks as an indicator of usability.

More advanced yet convenient tools for objectively measuring usability are also available. Jain et al. (2012) suggest using fuzzy logic, which is a probabilistic reasoning system. It involves representing facts with truth values that are not discrete (i.e. either 0 or 1) but with values that can vary (i.e. between 0 and 1 inclusive). It could potentially represent complex systems using simple rules. This would seem appropriate because not only is it a simple technique but also it is more befitting because unlike cost and time, quality cannot be determined precisely. Also, user requirements tend to change so the system of evaluation must also be able to adapt.

4.8.3 Usability evaluation in e-learning

The evaluation of usability in the case of e-learning software is considered to be difficult due to constraints of time, cost and know-how as a consequence of which there is a lack of studies on e-learning software usability (Srivastava et al., 2009). As pointed out by Feldstein (2002), there is no standard means to evaluate the usability of e-learning software. In these circumstances, it is normal for only the level of user satisfaction to be ascertained, so other important aspects of usability tend to be ignored, hence the need for

better understanding of usability with respect to e-learning software.

4.9 The cost factor in usability

4.9.1 Cost of implementing agile and traditional methods

A notable trend in the software development industry with the aim of reducing costs is offshoring, in which all or part of the software development process is given over to and undertaken in another country at a lower cost. Offshore development can be based on either traditional or agile approaches. The latter emphasise face-to-face communication and physical proximity, which are challenged under this arrangement, but as shown by Fowler (2003), it is still possible to apply agile techniques in offshore settings. He also compared these with plan-driven methodologies. The main software house was located in Melbourne, Australia and several projects over a period of two years for the comparison were undertaken by teams in Bangalore, India. Integration was achieved using distributed continuous integration methods and requirements were communicated using acceptance tests, both practices adopted from XP, face-to-face communication was supported by sending ambassadors from each site to the other, meetings were kept short but regular as were iterations, and teams were separated by functionality rather than activity. The greatest challenges were due to cultural differences so extra effort had to be put into dealing with them. Another interesting modification to typical agile methods was recognition of the importance of some documentation instead of downplaying this. Documentation was seen as a valuable substitute for the reduced face-to-face communication despite the impact of increased costs. In addition, collaborative tools such as wikis were also used.

As far as costs are concerned, rates are usually significantly lower for offshore development, but the overall return on investment is more important. The experience of Fowler (2003) highlighted the importance of considering productivity differences over salary differences, the problem of certain costs actually increasing, and risks arising that could offset the price differential. Traditional plan-driven methods cope well in offshore settings, mainly due to the extent of documentation, but agile methods proved to be better overall despite the difficulties with culture and communication. The lesson that can be learned from this is that at least some limited planning and documentation, as in UgCD, could be especially useful in such situations as offshore development. Secondly, the cost factor may not always be as important as quality factors.

4.9.2 The cost factor under UgCD

Bringing about improvements in the quality of software, and particularly making it highly usable is desirable, but in practice there are time and cost constraints that have to be considered. Theoretically at least, incorporating usability testing and elements to improve usability would increase the costs of a software project because they constitute an additional aspect of design. The extent of ensuring usability therefore needs to be weighed against the likely costs that would be incurred.

One way of ensuring usability is to undertake eyetracking research, but the cost of doing so remains very high (Nielsen & Pernice, 2010: 40), not to mention that it is also questionable as to its usefulness. Of the more common usability evaluation methods, HE and CW are the most cost-effective (Hwang & Salvendy, 2010), but each has its limitations as mentioned above. Whichever method is used, an important consideration in calculating costs during usability evaluation is the number of evaluators or users that will be tasked to evaluate the software's usability. The sample size is usually determined by estimating the discovery rate of the problem across participants. This rate is a quantitative measure that shows how effective a certain usability evaluation method is.

No studies were found that have compared UrCD and UgCD in terms of cost. However, to give an idea of how incorporating design elements to improve usability impacts on the costs under UgCD, a software development projects is described which was reported in Constantine & Lockwood (2002). The project was to develop a classroom information management system for which an agile UgCD was adopted, and which involved identifying essential use (or task) cases and user roles. The programming side of the project, which was undertaken using an agile method, was done in parallel to the visual and interaction design. Several ad-hoc accommodations had to be made during the process. That is, several ambiguities, omissions and irregularities had to be overcome, and a major design revision had to be made after an inspection was carried out for usability defects. The goal of ensuring usability was important because the design was to be made immediately usable by teachers with little or no prior training and with little time for lesson planning. This required, for example, maximising screen real estate, minimising modal behaviour, and allowing for simple and direct switching between different contexts.

One of the important lessons learned from this project was that improving usability is costly as well as risky, even when using streamlined methods. In particular, increased costs

becomes an issue with poor designs that lead to user errors being committed. Post-development costs can also be very high for designs plagued by usability problems. At the same time, this project also showed the usefulness of UgCD because neither traditional heavy plan-driven methods nor lightweight agile methods would have been able to provide for an upfront design of the visual, navigation and interaction schemes along with details of the expected roles and tasks. An early understanding of this information was necessary so that a quick to use, flexible and easy-to-learn system could be developed.

Agile methods in general are better suited for situations in which requirements can change quickly, and where the cost factor is not as significant as speed factors. This is because they tend to prioritise speed of development and responsiveness over both quality and cost (Baskerville et al., 2003), but often, faster responses may be more important than merely saving money. In fact, agile methods such as XP recognise documentation as a cost and that incomplete documentation is usually more cost-effective (Cohen et al., N.d.: 34). This attitude could explain the aversion to detailed planning by agile practitioners who attempt to prevent a situation of 'analysis paralysis'. Also, in theory at least, it is thought that bringing about continuous improvements in the design make subsequent iterations less costly (Abrahamsson et al., 2003: 15). On this basis, some surveys suggest agile methods are able to not only deliver quickly, but also improve quality as well as save money. For projects in which the cost factor is significant though, a more Value Based Software Engineering (VBSE) process may provide a better solution, such as described by Boehm (2003). In this study, an 'Earned Value Management System' technique of VBSE was shown to be most useful for controlling not only costs, but also the schedule and progress of a large software project.

Chapter Five

Requirements Analysis, Gathering and Capturing

Chapter 5: Requirements analysis, gathering and capturing

5.1 Requirements, their importance, and requirements capture

5.1.1 Definition of requirements

A requirement is a statement of the expected characteristics of the system, i.e. of what it is expected to do, or how it is to perform. Usually, the goal is to fulfil user needs. As such, it is "a necessary attribute in a system, a statement that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility to a customer or user" (Young, 2003). The term 'requirement' is also defined by the IEEE standard 610.12-1990 as 'A condition or capability needed by a user to solve a problem or achieve an objective; that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents'. This standard specifies the nature of a requirement as a certain condition or capability, and as with Young's definition, includes mention of the purpose of identifying and specifying a requirement.

5.1.2 Importance of gathering requirements

At least some allowance for determining requirements is important because it also helps to better understand the software development project at hand, select an appropriate developmental method, and to satisfy its requirements (Sommerville, 2007). In practice, many software developers tend to allocate insufficient time and effort, if at all, on requirements gathering, but it is important as it provides the basis for the development work to follow (Young, 2003). This approach reflects the growing need for adaptable software development methods in line with specific needs, as sustainable and successful software projects require elements of both discipline and agility (Boehm & Turner, 2003: 23). The usefulness of requirements gathering is reflected in the one-third drop in failed software projects between 1994 and 2006 reported in the Standish Group's Chaos Report of 2006 attributed to improved communication of requirements (Pohl & Rupp, 2011).

5.1.3 Requirements capture

This study specifically focuses on requirements capture, i.e. the ability to capture or correctly identify or foresee the immediate and foreseeable requirements of the software during its planning phase that are of concern to its usability. Notably, the aim of this study

and the research questions use the term 'Requirements Capture' and not 'Requirements Gathering', 'Requirements Analysis' or 'Requirements Elicitation', although these terms are used loosely elsewhere in this paper to mean the same thing. This use of all the aforementioned terms interchangeably is reflected in the literature in general. For example, Sofia (2010) described 'Requirements Engineering' as a systematic requirements analysis process and mentioned that terms such as 'Requirements Gathering', 'Requirements Capture' and 'Requirements Specification' are used loosely in referring to requirements engineering.

On the other hand, a few writers have attempted to define the terms differently. Brooks & Jones (1996) for example, suggested 'Requirements Capture' denotes a passive role of clients while 'Requirements Engineering' denotes a more mechanical process. However, these two terms were not distinguished from other similar terms, such as 'Requirements Gathering', and all were described as passive in an argument that was for a more cooperative approach in requirements analysis whereas the degree of passivity or cooperation with clients is determined by the actual methods used to gather requirements and the overall approach.

As for distinguishing between the different phrases, this study takes the view that Requirements Gathering is a more general term, Requirements Engineering implies a more systematic process, and Requirements Elicitation focuses on the involvement of users in the process. While these are used interchangeably with Requirements Capture, the term Requirements Capture is used herein specifically to acknowledge that requirements pre-exist but await being identified as such. This underlies the aim of this study which suggests UgCD is better able to identify requirements that are important for software usability that other methods may miss. That is, it implies UgCD is better able to target and capture usability requirements. Moreover, a requirements capture situation reduces ambiguity about particular requirements (Coughlan & Macredie, 2002: 12), which suggests the requirements have not only been identified but identified correctly. Hence, the justification for using the term Requirements Capture in specific contexts.

5.2 Types of requirements

5.2.1 Different types of requirements

The various types of software requirements can be divided into *functional* and *non-*

functional requirements. From another perspective of usability, some requirements are those that directly impact on the usability of a software, and are hence termed as *usability requirements*, whereas others are not. Hence, a distinction can also be drawn between *usability requirements* and *non-usability requirements*. This distinction is necessary to be made because this research focuses on capturing usability requirements. Besides these classifications, certain requirements can also be described as *conceptual* and *organisational*. Conceptual requirements are of an abstract nature and relate to the kind of services provided by the software, which may be subdivided into learnability and practicability requirements (Manza, 2010). And, organisational requirements pertains to any particular needs of the organisation in which the software is to be used. A distinction can also be drawn between *user requirements* and *system requirements* (Sundar, 2010: 23). The former category pertains to the services that the system provides together with its operational constraints, whereas the latter details the system services. Other requirement types mentioned in the literature include *performance requirements*, which concern how well a system is expected to perform, and which may be measured in terms of quality, quantity, timeliness, coverage or readiness, *design requirements*, which relate to the design of the software, *derived requirements* that are either implied or transformed from higher level requirements, and *allocated requirements*, which are established by dividing a high-level requirement into several lower-level requirements (Sofia, 2010).

5.2.2 Functional requirements

Functional requirements define precisely what is to be built, so they are precise about the expected behaviour though they may not precisely identify the possible implementation choices (Franke, 2006). As such, they pertain to processes that the system is expected to perform and the information it is expected to contain. Non-functional requirements on the other hand, define precisely *how* the application is expected to perform. As such, they typically pertain to operational, performance related, interface, and security aspects, as well as other behavioural properties, such as cultural and political related. In specifying either kind of requirements, the implementation or environmental details that restrict design choices need to be taken into account. For example, the software could be required to run on a particular system, database, use a particular type of code, and adhere to specific standards.

The following are examples of functional requirements:

- Requirements related to the processing that will be required to be carried out by the system
- Details of the system inputs and outputs
- Descriptions of data that will need to be held in the system
- Statements of services the system should provide, how it should react and behave in certain situations (Sundar, 2010)

System inputs my be obtained, for example, from interactions between people, paper-based forms, documents, or from other systems, and system outputs may be obtained from documents, reports, screen displays, or as transfers to other systems.

The following are examples of non-functional requirements:

- Performance criteria, such as acceptable response times while processing or retrieving data
- Expected data volume
- Security related considerations

In short, functional requirements deal with such aspects as tasks, activities, screens, data flow, inputs and outputs whereas technical requirements deal with such aspects as availability, reliability, performance, backup, recovery, archive and security (Harschnitz, 2011).

5.2.3 Usability requirements

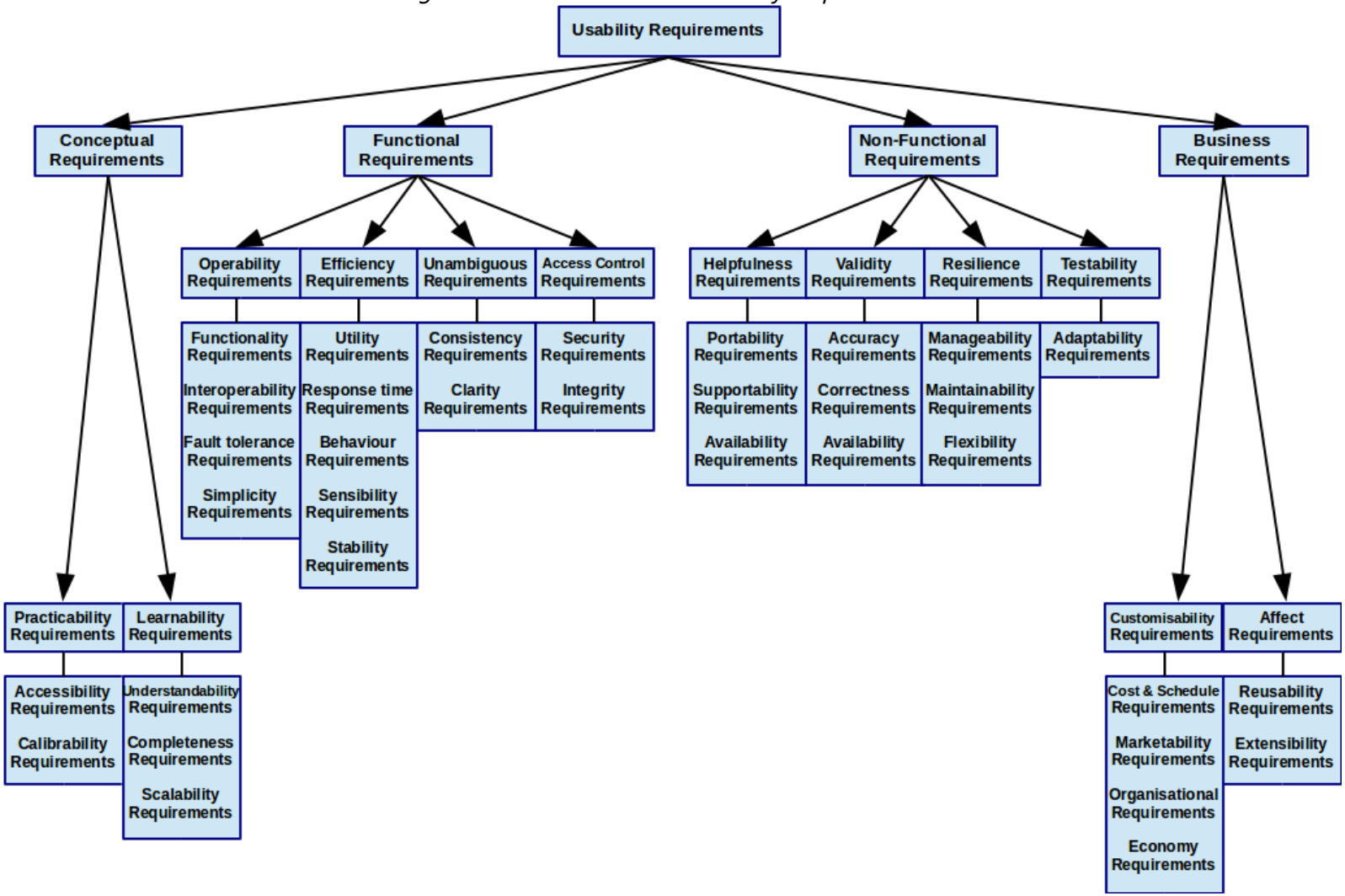
As this study is concerned with UgCD with its focus being on usability, another category of 'usability requirements' needs to be described as well. Bennett (2004) describes them as those requirements that ensure “a good match between the system that is developed and both the users of that system and the tasks that they will undertake when using it” (p. 121). These requirements are therefore those that enable the system to give users access to the software and the system tasks. In contrast with other user requirements mentioned in 5.2, they help to prioritise usability work and ensure usability goals are achieved, especially concerning efficiency and satisfaction (Hammond et al., 2002: 141).

Findings from interviews reported in Hirose (2001) revealed that such requirements tend to

be either not documented at all or get documented very late in the SDLC. Even awareness of usability standards are almost non-existent, as found in research on usability awareness in Malaysia (Ian et al., 2011). Given the importance of usability established earlier in section 4.3, assuring that a software is highly usable would require systematically identifying usability requirements from different views before the software is developed because usability requirements encapsulate usability goals (Manza, 2010). Moreover, analysing usability requirements at the beginning increases the likelihood of the software project being successful in terms of simultaneously satisfying user goals and objectives. As for requirements in general under RE, the usability engineering process necessarily involves a usability assessment. This helps to ensure the existence of usability in the developed system, and is achieved through measuring user performance and user satisfaction. Again, as stressed by Hussey et al. (2001) a usability assessment must begin at the conception stage of the SDLC in order to assure usability.

Usability requirements can be either functional, non-functional, conceptual, or organisational. The only criteria, as mentioned above, is whether they prioritise usability and ensure usability goals are met. Manza (2010) made an interesting multi-layered classification of usability requirements, in which the four main divisions are of conceptual requirements, functional requirements, non-functional requirements, and business requirements. These and their further subdivisions are illustrated in Figure 9 below. For clarity, the first two layers of divisions are listed in the table further below.

Figure 9: Classification of usability requirements



Source: Manza, 2010, p.581 (adapted)

Table 7: Main classifications of usability requirements

Usability requirement classifications of the top two tiers listed in Manza (2010)			
Conceptual requirements	Functional requirements	Non-functional requirements	Business requirements
Learnability Practicability	Operability Efficiency Access Control Unambiguous	Helpfulness Validity Resilience Testability	Customisability Affect[ive]

5.3 Requirements engineering, lifecycle and analysis

5.3.1 Requirements engineering

There is a growing interest in Requirements Engineering (RE) in general along with a recognition of the importance of eliciting appropriate software requirements during the early software development phase in recent years (Zhang, 2010). As such, the impact of RE, especially on the development of customer-oriented systems, cannot be ignored any longer (Pohl & Rupp, 2011).

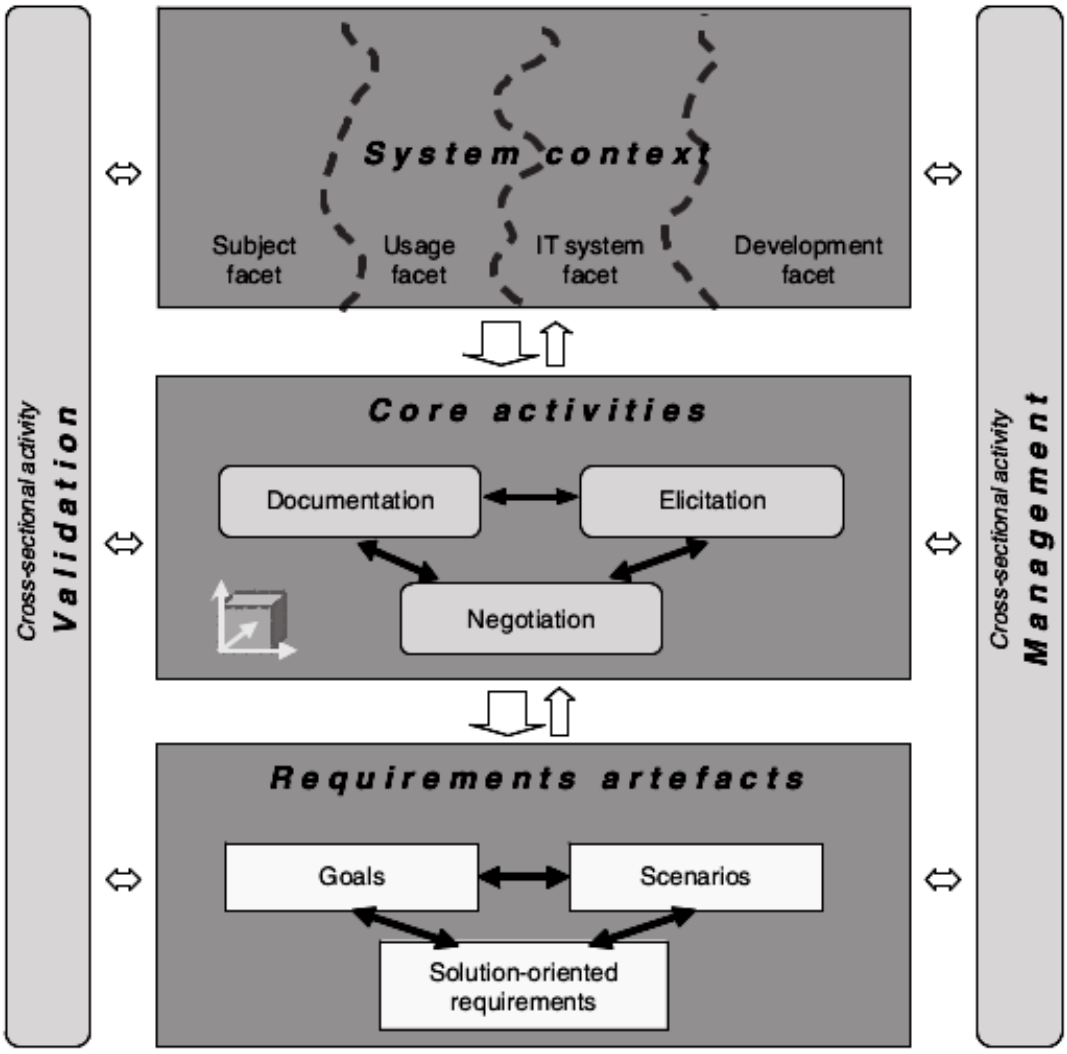
RE refers to “a systematic and disciplined approach to the specification and management of requirements” with specific goals in mind (Pohl & Rupp, 2011). This would contrast with an ad-hoc approach that is uncharacteristic of RE. The goals concern knowing the relevant requirements and documenting them appropriately. Zave (1997) termed these as 'real-world goals' in a more elaborate definition of RE, which also highlights their relationship with 'precise specifications' and their 'evolution' as well:

> *“Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.”*

According to Pohl (2010), the process of RE involves three core activities: elicitation, documentation, and negotiation; it involves distinguishing three essential types of artefacts: goals, scenarios, and solution-oriented requirements; is structured by the contexts of subject, usage, IT system and development facet, and is guided by two cross-sectional

activities: validation and management support. These four aspects of the RE framework are illustrated in Figure 10 below.

Figure 10: The requirements engineering framework



Source: Pohl (2010)

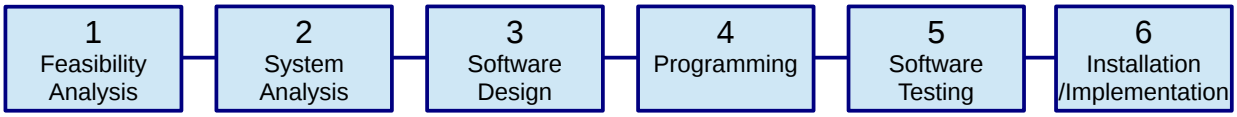
The field of RE makes use of the concept of a *stakeholder*, which refers to people or organisations that in some way or other impact on the software requirements. This is due to them being a major source for ascertaining requirements (Pohl & Rupp, 2011), and also given that requirements tend to be incomplete or 'fragmentally elicited' if they are not consulted (Macaulay, 1993). The latter justification was made under the view of software development as a complex social process rather than a mere mathematical or technological one. Stakeholders include not only the expected users of the software to be developed, but also others who are likely to interact with it or have an interest in it, as well as other

influential parties such as legal institutions. The social process view suggests that the process of 'requirements capture' and the whole construct of 'requirements engineering' are better seen as cooperative and dynamic rather than as passive approaches in software engineering.

5.3.2 Requirements lifecycle

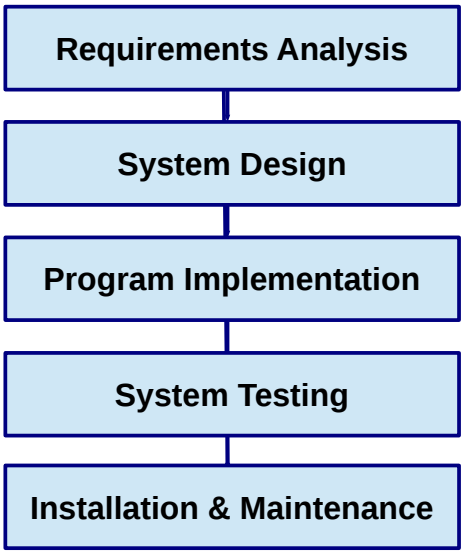
A typical SDLC (Software Development Life Cycle) process is shown in Figure 11 below. It usually beings with preliminary feasibility and system analysis phases in which the system to be constructed is analysed. This information is then used to design the software, after which the coding takes place. Usually, this is followed by testing of the software before finally implementing the solution by installing the software. Whilst all software development methodologies generally follow this typical pattern, they differ in the extent to which each phase is emphasised and the way in which each is carried out.

Figure 11: Typical software development life cycle



Another typical (linear sequential) Software Development Life Cycle (SDLC) is shown in Figure 12 below in which the first phase is specified clearly as Requirements Analysis. A SDLC specifies the order or sequence in which software development activities are carried out. However, agile methods attempt to gather requirements later in the cycle even after programming has commenced, as some requirements tend to change over time as the project develops during its life cycle from analysis through to design, programming and implementation. This distinction was pointed out in section 2.3.

Figure 12: Phases of a typical software development lifecycle



Source: Sabharwal (2008)

5.3.3 Requirements analysis phase

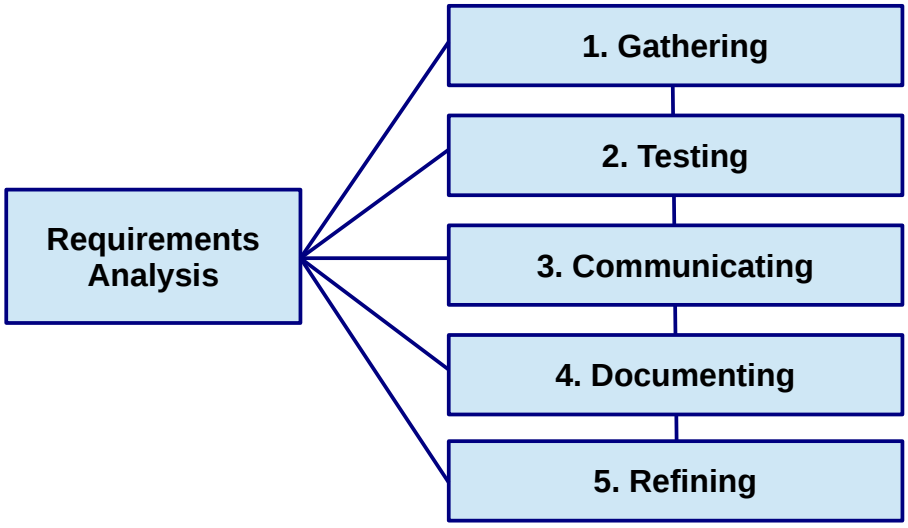
It can be seen from the above diagrams that requirements are usually ascertained prior to the design in traditional software engineering methods. Gathering or capturing requirements during this stage of software development is important because this is the earliest point in time that potential performance issues can be addressed while the architecture is still being formulated (Petriu, 2002).

This study focuses on the pre-programming stages of systems analysis and software design, specifically on the way in which requirements are captured, i.e. identified and refined. Requirements capture is part of the system analysis stage whereas any later refinements take place during the design stage. Two key processes in connection with requirements are elicitation and documentation. Elicitation involves using different techniques to obtain and refine software requirements (Pohl & Rupp, 2011), and documentation involves adequately describing the elicited requirements using either conceptual models or natural language (see section 5.3). Following elicitation and prior to documentation, the requirements gathered are usually also checked, for example, to ensure whether they are correct, complete, non-ambiguous, non-contradictory, etc. (Sofia, 2010). In summary, the three types of activities undertaken during a typical requirements analysis are as follows:

1. Requirements elicitation/gathering: involves communicating with users to determine their requirements
2. Requirements testing: determining whether the requirements satisfy certain criteria
3. Requirements documentation: recording of requirements in a certain form

Some methods however, including agile methods with UgCD included, continue the process of requirements analysis after documentation to *evolve* or *refine* them further during the design and programming stages of SDLC. Nuseibeh & Easterbrook (2000) describe the ordering as (1) elicitation, (2) modelling/analysing, (3) communicating, (4) agreeing, and (5) evolving, but using the terms 'testing', 'documenting' and 'refining' instead, as adopted in this study, the typical sub-phases of the requirements analysis phase can be illustrated as shown in Figure 13 below.

Figure 13: Sub-phases of the requirements analysis phase



5.4 Requirements gathering and analysis frameworks

5.4.1 Gathering requirements

Along with making models, gathering user requirements is a key feature of UgCD during both the pre-programming and programming phases. Task modelling is used to aid gathering requirements during the design stage, and further refinements are made afterwards as is typical of agile methods. The procedure is described further below in

section 5.6. As for the pre-programming stage, this is typical of traditional methods in which considerable time is allocated for determining requirements before designing and programming the software (Parsons & Lal, 2006). In contrast, pure agile methods are generally considered to be inadequate at handling requirements (Kavitha & Thomas, 2011), although SCRUM for example, does make some allowances for ascertaining user requirements by having an on-site customer, and user stories are used to gather changing requirements. Rather than handle requirements during the initial planning or pre-programming phase, agile methods are better able to handle changing requirements due to their incremental development approach. The neglect in gathering initial requirements under agile methods is due to the priority given to speedier and responsive software development (Baskerville et al., 2003).

The actual methods used to gather or elicit requirements depend on the guiding methodology (see 5.4.5). Common methods include one-to-meetings, group interview meetings, facilitated sessions, questionnaires, observations, document analysis, and prototyping. In addition, requirements may be sourced from such documents as recordings of observations and interviews, descriptions of problems, minutes of meetings, etc., which may not fit into a UML framework (Bennett, 2004). Nuseibeh & Easterbrook (2000) presented a broad classification of the different requirements elicitation techniques, which is shown in Table 8 below. In this scheme, UgCD makes use of the fourth type of method initially followed by the second for later refinements, i.e. modelling and group elicitation.

Table 8: Classification of requirement elicitation techniques

Classification of method/technique	Description	Examples
Traditional	A broad class of generic techniques	Questionnaires, interviews, analysis of existing documentation
Group	Aim to foster stakeholder agreement while exploiting team dynamics	Brainstorming, focus groups, consensus-building workshops
Prototyping	Used when there is great uncertainty about requirements or if early feedback is needed	Mockups
Model-driven	Provide a specific model to drive the elicitation process	Scenarios, rich pictures
Cognitive	Originally developed for knowledge based systems	Protocol analysis, laddering, car sorting, repertory grids
Contextual	Emerged in the 1990s as an alternative to traditional/cognitive techniques	Ethnography

Source: Nuseibeh & Easterbrook (2000) (outlined and tabulated)

The next sub-section details the IEEE specifications for developing an SRS and discusses requirements testing generally after they are elicited and before they are documented. The next section then explores the various possible requirements gathering frameworks that may be used to guide the overall process of gathering and documenting requirements.

5.4.2 Requirements gathering recommendations and issues

The IEEE makes several practice recommendations for developing a good SRS (Zelenty, 1998). The purpose of the recommendations are to ensure software users are able to describe what they wish to obtain accurately, to aid developers to understand exactly what users want, and so on. An SRS is considered to be good if it has the following identified characteristics:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Correctness is achieved if all the stated requirements are met by the software, unambiguity if each requirement has only one interpretation, completeness if all significant requirements are included, consistency if it has internal consistency, importance/stability ranking if each requirement includes an identifier indicating its importance/stability, verifiability if each requirement is verifiable, modifiability if its style and structure permits any modifications to be made easily, and traceability if its origin is clear.

This study is primarily concerned with the first and third criteria of correctness and completeness where correctness is referred to herein as preciseness. Testing will be done to

show that UgCD can provide a more complete and precise list of usability requirements. The selection of these two criteria is reasonable given that there is an acknowledgement of obstacles to realising a completed system, which are believed to be due to both technical and social reasons (Coughlan & Macredie, 2002). Most studies suggest a key factor in problems in system design is communication breakdowns. For example, Macaulay (1996) showed the following to be leading causes:

1. Poor communication between people
2. Lack of appropriate knowledge/shared understanding
3. Inappropriate, incomplete or inaccurate documentation
4. Lack of a systematic process
5. Poor management of people/resources

Although the first cause is directly related to communication, the second and third are also due to communication issues so communication is an important issue in the field of RE. Other studies have also highlighted issues that concern communication, such as Beckworth & Garner (1995), Al-Rawas & Easterbrook (1996), and Sutton (2000). In Al-Rawas & Easterbrook's (1996) study, the communication difficulties were narrowed down to restrictive notational languages, ineffective communication channels and social and organisational factors. Notably, UgCD relies on a well-established notational system, as used in Jacobson's use cases, and communication is strengthened through its use of JAD sessions and JITR (see 5.5.2 and 5.5.4 respectively).

5.4.3 Documenting requirements

Requirements can be documented (or recorded) in a number of ways, but they would be done either in a manual, or automated manner in digital form, or a combination of both. That is, requirements can be either recorded on paper or entered into a computer. A specially prepared Requirements Analysis Sheet (RAS) is usually used for stating the captured requirements on paper, and model entities and requirements entities are usually mapped to clarify relationships and facilitate traceability (Grady, 2010: 172). The output document of the requirements analysis stage of the SDLC is also referred to as a Requirements Specification Document (RSD) or Software Requirements Specification (SRS). Complex software projects would require Computer Aided Software Engineering

(CASE) tools to be used. They could also assist in checking whether the captured requirements have been actually met afterwards.

Whichever approach is used, the requirements are usually recorded in the form of either text, diagrammatically, or again a combination of both. Regardless, it is important that they are recorded in a consistent manner, for which purpose the process should adhere to some set of standards. UML provides one way of producing systematic and standardised visual models through its specification of graphical notations, as described earlier in section 2.4.1.

5.4.4 Requirements gathering frameworks

Major requirements gathering frameworks include the Karl Wiegers methodology, User Centred Design (UCD/UrCD), interaction design, and Object Oriented Analysis/Design (OOA/OOD) (Franke, 2006).

According to Franke (2006), in UrCD, software features and functions are driven formally based on use cases, scenarios or scripts although the relation to value is informal. The reliance of UrCD methods on scenarios was also noted by Hudson (2001) but it was also noted that use cases are actually associated with informal UML, not UrCD, to be precise. Another method that is more focused specifically on scenarios is called Scenario Based Design (SBD), which was described briefly in 2.7.2. It is thought that scenarios of Human-Computer Interaction (HCI) can help to better prepare for certain technical challenges that could arise (Carroll, 2000).

In interaction design, the primary users of the system are identified and solutions are then specifically designed for them, and the framework involves the application of psychological and human factors. The OOA framework uses the UML based technique for presenting analysis findings. It is reliant on supporting documents, and involves the construction of two object models, namely one that describes the system and one that describes the solution. The Karl Wiegers methodology is a framework used to describe requirements hierarchically. It therefore differs from UrCD in that the goals underlying the use cases are articulated formally. These aforementioned key features are summarised in the table below.

Table 9: Key features of four requirements gathering frameworks

Karl Wiegers Methodology	**User Centred Design**
• Requirements are described hierarchically	• Software functions/features are driven formally based on use cases, scenarios or scripts; • Informal relation to value
Interaction Design	**Object Oriented Analysis/Design**
• Identification of primary users for whom specific solutions are designed; • Application of psychological and human factors	• UML used to present analysis findings; • Object models are built for the system and solution

Source: Franke (2006) (adapted)

5.4.5 Specific requirements gathering frameworks and methods

Common methods used to gather requirements are numerous. These were mentioned in 5.4.1. The actual methods used however are determined by the guiding framework or methodology that were introduced in the previous sub-section. This sub-section explores specific requirements gathering frameworks and methods in more detail by describing which methods are used by a select number of frameworks.

As regards guiding methodologies or techniques, popular ones include the Rational Unified Process (RUP), Joint Requirements Planning (JRP) (Kulak & Guiney, 2012), Joint Application Design (JAD), Use Case, and Unified Modelling Language (UML). A select number of these along with some others are briefly described below, but JAD, ACD, JITR and SBD are described further on in section 5.5 due to their similarities with UgCD. As for tools and instruments used for gathering requirements, these include, for example, templates, base documents, checklists, workshops, signoff, and change process (Harschnitz, 2011). Diagrams are particularly common artifacts in OOA/OOD generally.

The LOTOS-based approach to capturing requirements for example, involves raising questions to identify the system's users, its context and boundary, the kind of functions it can rely on, the type of data inflows and outflows and their content and structure, the functions that the system will be expected to perform, the relationship between inputs and outputs, any system limitations, and so on (Habrias & Frappier, 2010: 235). These concerns are distinguished between the environment, interfaces, functionality, limitations, non-functional aspects and methodology. LOTOS is a formal algebraic based description

technique for capturing requirements used for describing systems (Amyot et al., 1999).

JRP is similar to JAD in that sessions are held to conduct user interviews simultaneously, but it differs in its focus on *what* the system will do whereas JAD focuses on *how* the system will work (Kulak & Guiney, 2012). A JRP session typically begins with discussing high level topics, such as critical success factors and strategic opportunities, and then moves onto identifying, documenting and prioritising functional and non-functional requirements. In the Abstract State Machine (ASM) method, the requirements capturing process leads to the development of ground models that are concise but precise high-level system blueprints.

The table below summarises the information on how requirements are actually gathered for the above mentioned and other requirements gathering methods, methodologies or frameworks for ease of comparison.

Table 10: Overview of how different methods gather requirements

Method /Framework	How requirements are gathered
ACD methods	**Activity modelling** is used for **systematic** organisation and **contextual** representation of tools; Focus is on user **activities** and on **tasks** to be performed.
ASM	**Ground models** are developed in the form of concise **high-level system blueprints**.
HCD methods	Focus is on software **users**.
Interaction Design	**Specific solutions** are designed for **identified primary users** based on **psychological** and **human factors**.
JAD	A **joint structured meeting** is held which focuses on ***how* the system will work**. User involvement is elicited using **dynamic group techniques**.
JRP	A **high level session** is held to **discuss critical success factors and strategy** before gathering requirements while focusing on ***what* the system will do**.
Karl Wiegers	Goals underlying use cases are articulated **formally** and requirements are described **hierarchically**.
LOTOS	Questions are raised to **identify system** users, functions, context, boundary, reliable functions, and **type, content and structure** of data in/outflows, etc.
OOD	**Object models** are built for system/solution and **UML** is used to present findings.
Participatory Design	Holding of **structured, facilitated interactions** between designers and users.
SBD	Uses **scenarios of HCI** specifically to overcome certain **technical challenges**.
UML	Use of **use cases** and represented using **graphical notation**.
UgCD	Applies **activity theory** with a methodological scaffolding; Uses **task modelling**, and **JITR** for further refinements; Focus is on **usability**; Similarities with JAD/ACD.
UrCD methods	Generally rely on **scenarios**.

The above mentioned framework and techniques can yield numerous requirements, so quality issues need to be considered in requirements gathering. According to Harschnitz

(2011), requirements are deemed to be effective if they are complete, specific, measurable, achievable, connected and signed off by clients. Furthermore, they should be prioritised by distinguishing between essential and optional requirements.

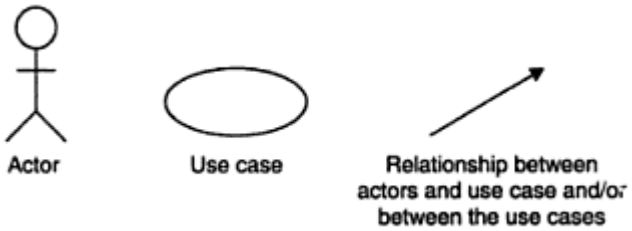
5.5 Constructing models of requirements

The section provides a brief overview of methodologies, models and frameworks that are either adopted or have been adapted for use by UgCD, namely use case models, task modelling, JAD, ACD, JITR, and scenarios.

5.5.1 Use case models

Use case models make use of what are called use cases in software engineering. A use case defines a procedure for the interactions between an actor or role and the system in order to achieve a certain goal. The main components of a use case model are shown in Figure 14. The role does not necessarily have to be carried out by a human, as it may also be an external system. A typical process for defining a use case involves identifying the users of the system, creating a user profile for each category of user, identifying significant goals for each role, structuring the use cases, and validating the users. In identifying the significant goals, the most significant role defines the system's 'value proposition'. The purpose of a use case is “to capture 'what' the system is, and not 'how' the system will be designed or built'” (Aggarwal & Singh, 2005: 51).

Figure 14: Components of a use case diagram



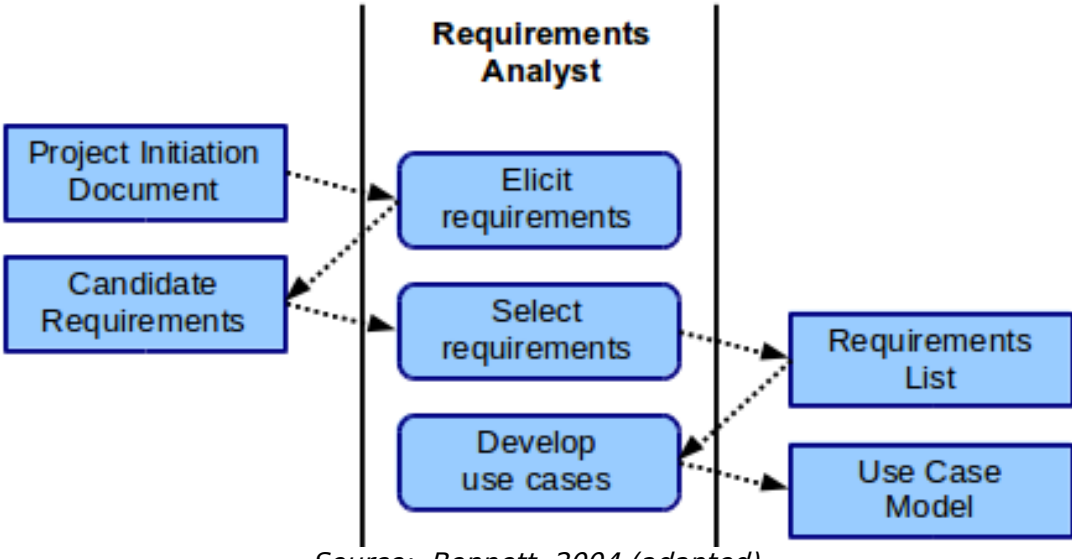
Source: Aggarwal & Singh (2005)

Traditionally, requirements have been identified using Jacobson's use case model for capturing functional requirements, and an additional list is then prepared of any supplementary requirements, i.e. those that are not identified through any use case (Bennett, 2004). The model devised is referred to as the requirements model, in which the

problem domain and user interface descriptions are also included. A use case specifies the sequence of actions that are to be performed by a system so as to yield a value to a particular actor that can be observed, and the purpose of the use case diagram is to show the relationships between actors and use cases, not the possible scenarios (Haumer, 2004). Notably, use cases are better suited for only documenting functional requirements, not necessarily non-functional requirements since their focus is on the system's functionality (Bennett, 2004: 133). Jacobson et al. (1999) themselves recommended the use case model only for functional requirements. Although there are other requirements capturing modelling approaches, use cases have the advantage of making it easy to view implementation decisions in requirements (McBreen, 1998).

Jacobson's Use Case Driven approach identifies three important system aspects, namely (1) actors, (2) use cases, and (3) system/sub-system boundary. Actors are the roles that users will assume. Use cases describe how the system will function from the perspective of users, or as stated by Rosenberg & Scott (1999), they describe units of system behaviour, which are then used to devise requirements that are like rules that describe the expected behaviour of the system. Use cases are therefore by nature *behavioural specifications*. A diagramming technique to show this is provided by UML, which is typically supplemented by more detailed descriptions of each use case. A simple illustration of the process is shown below.

Figure 15: Activity diagram showing requirement capturing activities



Source: Bennett, 2004 (adapted)

The textual descriptions mentioned above, i.e. *case descriptions*, describe in more detail the expected *interaction* between the actors (i.e. system users), and the various system functions considered to be of high level are detailed in the use cases. In other words, the use case itself details the interaction a user (or other system) will have with the system being designed for achieving a certain goal (McBreen, 1998). It is more formally defined in terms of its purpose of delivering “a measurable result of value to a particular actor” (McBreen, 1998: 4).

Use cases prove beneficial in that they highlight actor value, link with specific stakeholder goals, and so on (Haumer, 2004). On the other hand, devising use cases requires an expert systems analyst and care in avoiding pitfalls such as functional decomposition, over analysis, insufficient analysis mechanisms to develop abstracts from design decisions, etc.

In agile development, there is a similar sounding concept of 'user stories', so it is worth pointing out the differences. The two differ in terms of scope, completeness, longevity, details, and purpose (Cohn, 2004: 137-140). For instance, a user story is deliberately kept smaller in scope whereas a use case normally covers a much larger scope. Also, a user story is usually devised for a particular iteration and is not kept as a permanent artifact, as is done for use cases. A use case typically contains more details of the user interface, and is written in a format that is appropriate for both developers and customers whereas a user story is written for giving details of the release and iteration planning.

5.5.2 Task modelling

The idea behind modelling tasks is that people use mental models in performing tasks so the task models explicitly detail those mental structures to describe the cognitive processes (Bruning et al., 2007: 177). Although task diagrams are not part of UML, which offers activity diagrams for describing behavioural aspects of design, activity diagrams can be usefully applied for task modelling. Hollnagel (2003: 456) pointed out that the followng four representations in particular are directly relevant for the purpose of task modelling:

- The activity diagram, which can be used to describe the task flow in terms of roles, events and goals
- The collaboration diagram, which can help to show how different objects work together, such as arrows for the roles communicated

- The sequence diagram, which can be used to show the sequence of tasks as performed between roles
- The use case diagram for describing scenarios

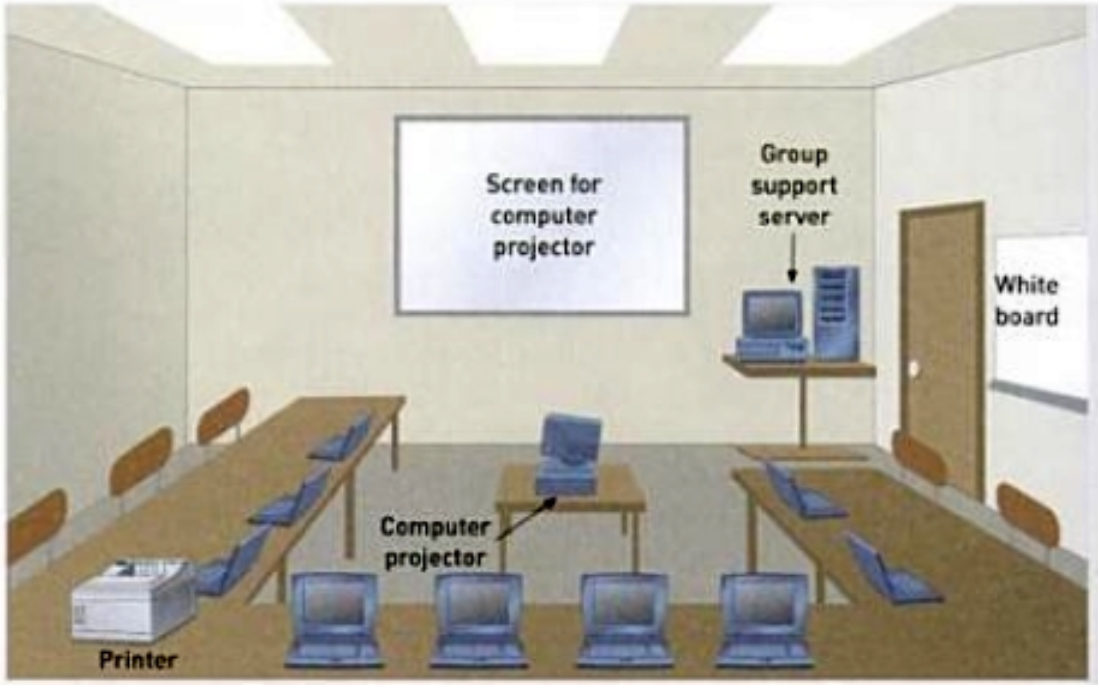
For working with UML however, some interpretations would have to be modified in order for it to be used for task modelling. These modifications are to treat states in an activity model as tasks, and objects as roles. Other notations have also been used for modelling tasks, such as CTT (Concur Task Tree) (Winckler et al., 2012: 299), but the advantage of using UML is the fact that it is an established standard and many tools exist for it, so it would be convenient for developers to work with.

5.5.3 Joint Application Design

As UgCD bears similarities with JAD, this design methodology will be described in detail as an example of how requirements can be gathered. As Carmel et al. (1992) described, JAD is a practitioner derived methodology for systems development that involves holding structured and facilitated meetings in which user involvement is elicited. The use of dynamic group techniques help to refine ideas. In fact, JAD was developed by IBM in the late 1970s specifically to arrange structure meetings to improve communications (Coughlan & Macredie, 2002) out of which such ideas can emerge.

A typical JAD facility in which JAD sessions are held in shown in Figure 16 below. The process involves defining the project and research with respect to various elicitation tasks, preparation for the JAD session based on the information gathered, and the actual group session itself (Coughlan & Macredie, 2002), which has become characteristic of JAD. Other methodologies such as Participatory Design also involve holding structured, facilitated interactions between designers and users, but JAD differs with respect to structure, degree of facilitators' control, and the mode of user involvement. Furthermore, JAD is focused on providing an accelerated design through obtaining high quality and comprehensive data. It is claimed that JAD results in increased quality, and at the same time a reduction in both costs and life cycle time.

Figure 16: A typical JAD facility



Source: Satzinger et al. (2008), p. 148

5.5.4 Activity Centred Design

Activity Centred Design (ACD) can be contrasted with User Centred Design (UCD or UrCD), which is a Human Centred Design (HCD) as the focus is on software users. Norman (2005) who himself coined the term UCD later suggested that too much attention has been given to users, and that there is a need to focus instead on the activities in which users engage, as well as on the tasks to be performed by those activities, in order for designers to be able to design better tools.

ACD makes use of activity modelling for systematically organising and representing contextual aspects of the tools used. This approach is based on activity theory for developing a conceptual framework to clearly depict how people use tools. UgCD applies activity theory by providing it with a methodological scaffolding, as described by Constantine (2006).

5.5.5 Just-in-Time Requirements

Just-In-Time Requirements (JITR) is an agile technique for gathering requirements which is undertaken during the actual development of the software. It is therefore best suited to deal with changed needs, changed priorities, etc. (Westfall, 2008: 166). User stories are

devised in order to capture high-level stakeholder requirements.

5.5.6 Scenario Based Design

Scenario Based Design (SBD) was described in section 2.7.2 as an example of another combined plan-driven and agile method, as is UgCD. The process of requirements development in SBD was described in a case study by Carroll et al. (1998) of a project that involved the creation of a virtual physics laboratory by a group of teachers and system developers. For this purpose, classroom scenarios were observed so as to assess needs and opportunities, as well as to envision potential scenarios in order to specify and analyse possible design moves. Design trade-offs implicit in the scenarios were evaluated using claims analysis. The requirements development process progressed continuously through stages from the initial high-level functional requirements of clients to a set of requirements that incorporated more non-functional requirements.

5.6 Requirements gathering under UgCD

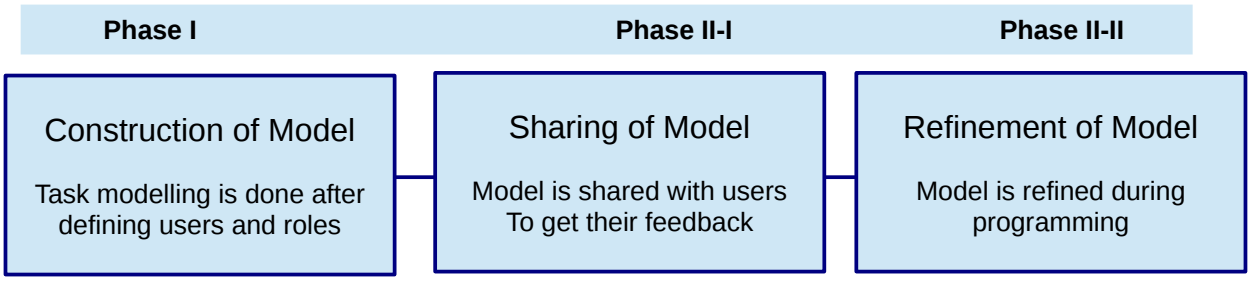
5.6.1 Outline of process

Under UgCD, an exploratory modelling process to identify roles and tasks is undertaken to ascertain user requirements (Constantine & Lockwood, 2001). This is an adaptation of Jacobson's Use Case driven approach mentioned earlier in section 4.6. Notably, Constantine's method defines essential use cases and distinguishes these from real use cases. Essential use cases are those that define the 'essence' of a use case whereas real use cases provide the concrete details of the actual design for each use case.

An essential use case is expressed in terms of user intentions and system responsibilities. The process involves raising questions and identifying potential areas for investigation through constructing the provisional user role and task models based on background information and knowledge available at the time. The focus is on prioritising usability through modelling task cases rather than use cases, and this consideration of usability increases the chances of ensuring usability of the final system (Juristo et al., 2003). Any omission, irregularities or ambiguities about the requirements are then dealt with during the frequent meetings and continuous consultations that are characteristic of the pattern known as Just-In-Time Requirements (JITR). This could take the form of, for example, sharply focused enquiries, investigations, surveys or observations.

The modelling process in UgCD therefore undergoes three distinct phases, which are illustrated in Figure 17 below. It begins with a construction of the model (task modelling) after first defining users (or actors) and specific user roles, and this model is then shared with users to get their feedback during the main meeting. These two phases are plan-driven in nature because they take place prior to programming. In the subsequent agile phase, the model is then refined while the programming is in progress during the several additional meetings and consultations as considered necessary. Both feedback and refinement are considered to be very important in UgCD. Clearly, UgCD is a methodology that is reliant on models to guide the entire design process, which includes user role models and task models.

Figure 17: Basic process of modelling in UgCD



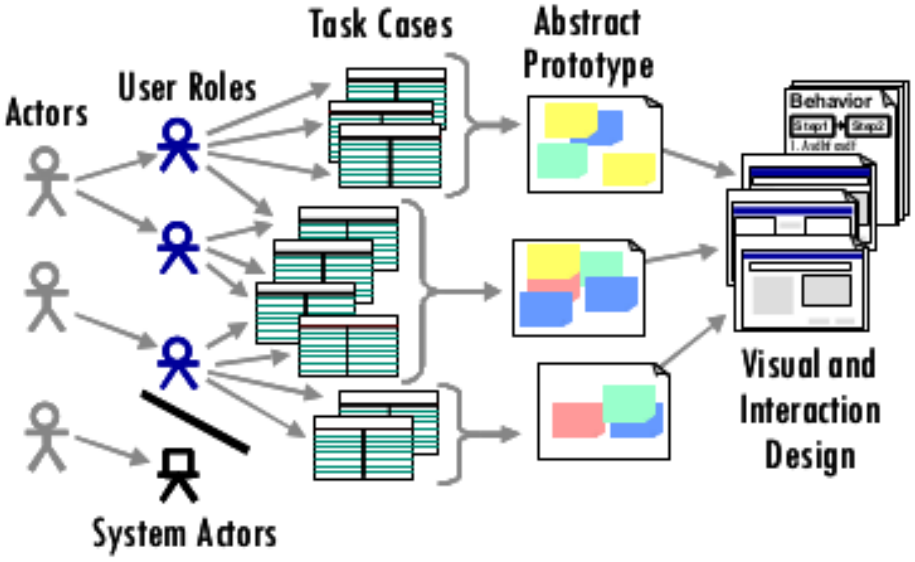
In summary, requirements gathering under UgCD has the following main characteristics:

- Task modelling adapted from Jacobson's Use Case driven approach
- A special meeting is held to conduct a requirements dialogue
- Approach has similarities with Joint Application Design (JAD) and Activity Centred Design (ACD)
- Further refinements are made using JITR
- Focus is on usability throughout

5.6.2 Detailed description of process

The logical process followed by UgCD during analysis and design is shown in Figure 18 below. The UgCD process is now examined in more detail.

Figure 18: Logical process of UgCD



Source: Constantine & Lockwood (2001), p. 6

UgCD involves an in-depth analysis and modelling during the system analysis stage, i.e. prior to the design stage, of the following three elements:

(1) The actors or users, who in the case of e-learning software will be mainly teachers, students and course administrators;

(2) User roles that define what each user will be able to do; and

(3) Use or task cases in which a list of tasks will be made and the requirements for each will be identified.

The types of users and the roles of each user is ascertained first before identifying the use or task cases pertaining to each user role, some of which may overlap. Each task case is then examined in turn in order to identify what capabilities the software will be required to be able to perform. The role and task models then help to develop abstract prototypes which are then combined to make the design of the software prior to commencing with the programming.

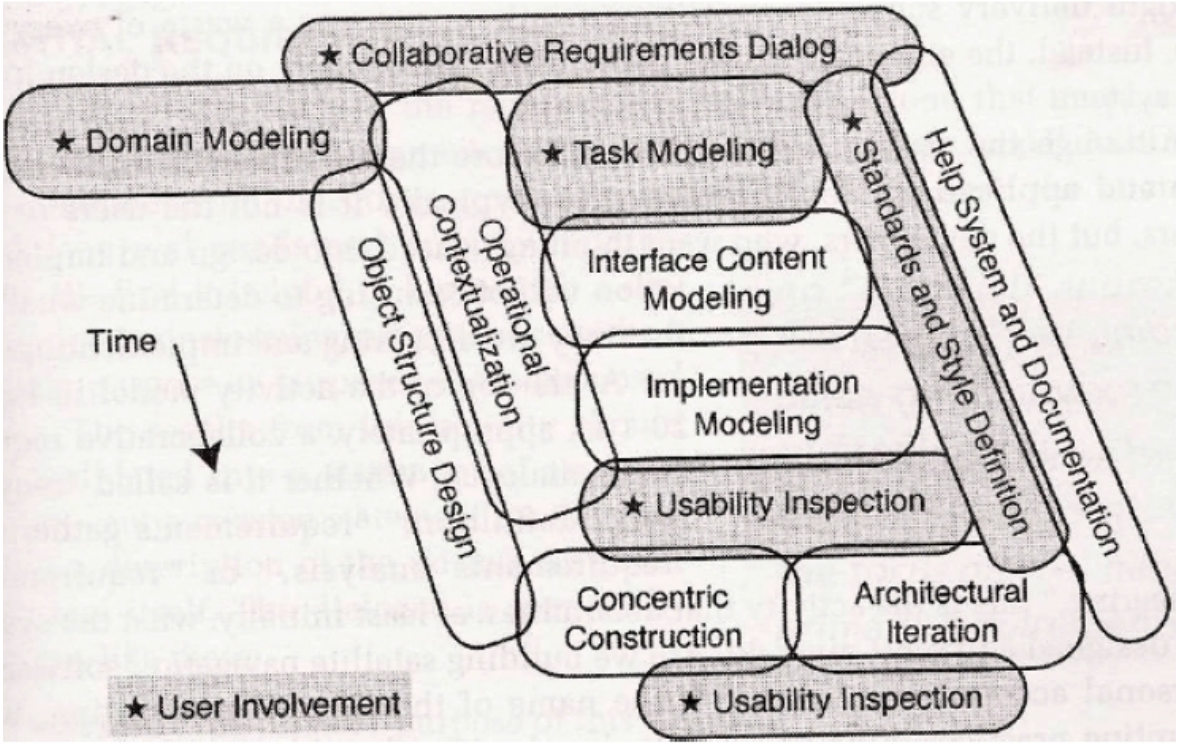
In UgCD, a user role is understood to be “an abstract collection of needs, interests, expectations, behaviors, and responsibilities characterizing a relationship between a class or kind of users and a system.” (Constantine, 2011, p. 79). This is substantially the same as

the notion of an actor in object-oriented software engineering introduced by Jacobson et al. (1992) except for one important difference: In UgCD, an actor does not include any nonhuman systems. As for the role model, this is merely a list of the expected user roles that are to be supported by the system, and each is described in terms of the aforementioned collection of items, i.e. needs, interests, behaviours and responsibilities that characterise and distinguish that particular role (Constantine, 2011, p. 80).

As mentioned earlier, the purpose of the task modelling, which is likely to be the most detailed of the three elements, is to help capture the requirements necessary to ensure and maximise software usability. In UgCD, the process of requirements gathering is seen as an activity “that determines, at least initially, what the system to be designed and built must do” (Constantine, 2011, p. 486). To aid in capturing the requirements for each identified task case and for refining the requirements later, UgCD makes use of a number of techniques, which include sharply focused enquiries, investigations, surveys and observations. In practice, this activity of gathering and analysing requirements tends to be subtle and sophisticated as it requires skills, good tools, and also sensitivity. It is neither simply a process in which requirements are ascertained from users, nor a matter of telling them; rather, it takes the form of a dialogue in which there is mutual exploration and negotiation until a consensus is reached between the developers and users.

As such, it is apparent that the requirements gathering process in UgCD is highly collaborative as there is significant user involvement. The illustration below shows this as a 'Collaborative Requirements Dialog' component of the UgCD activity model. This feature distinguishes UgCD from UrCD as UgCD is more user-involved rather than just user-centred. In UgCD, the end-user community is drawn in to the development of their software to engage in certain “tightly focused activities where and when their participation is likely to yield the greatest payoff for the project and all its constituents” (Constantine, 2011, p. 485). In order to facilitate the collaboration, the meetings take place face-to-face, the approach is conversational rather than inquisitive, and they are organised around specific aspects or topics in a way that would be conducive for a cooperative atmosphere.

Figure 19: UgCD activity model



Source: Constantine (2011), Software For Use, p. 485

Moreover, UgCD takes an 'essential' approach to gathering requirements in which intention is emphasised over action, and the attention is on goals and objectives so as to focus on actual needs and key requirements (Constantine, 2011, p. 487). This usually commences with a discussion or brainstorming session focused on the actual purpose of the system. A statement of essential purpose is then devised to describe the raisons d'etre of the system. The dialogue is sparked by questions pertaining to the system's primary purpose, how it is expected to be used, what is expected to be accomplished, and the reason for doing particular tasks. This dialogue typically takes place in a setting in which both developers and users face each other, in which a jointly prepared agenda has been distributed in advance, and in which a neutral facilitator oversees the interaction to ensure everyone is involved (Constantine, 2011, p. 491).

The purpose of the face-to-face meetings and discussions is to gather information, build the model, negotiate, and to approve and validate the gathered requirements. It is also pertinent to note that a distinction is drawn between needed functions on one hand and desired ones on the other. Also, the participants that are chosen are sought carefully for their potential knowledge, commitment and cooperation, and where possible the same people participate

in each meeting. The agenda provides the context which is necessary for facilitating collaboration. Four types of information are of concern during the requirements gathering process (Constantine, 2011, p. 489):

1. Function – basic capability of system
2. Form – realisation and appearance of functions within the system
3. Criteria – desired system characteristics or attributes
4. Constraints on acceptable/possible solutions

As a simple example of a requirement, if one of the many tasks is for the software to identify the user, then the main requirements will be to request and verify the identity of the user, i.e. the software will have to be designed and programmed in such a way that it will be capable of requesting and verifying the identity of the user. This is summarised in Table 10 below. In practice, and particularly for e-learning software, there are likely to be several user tasks for each of which the requirements will have to be identified, which would result in a long list of software requirements. Notably, UgCD provides a very systematic way of establishing requirements. This is seen as having many advantages, of which the most important is the potential to prevent 'creep' and 'leakage' of requirements (Constantine, 2011, p. 490). Requirements creep refers to expanding the system beyond what was initially agreed upon, and requirements leakage refers to additional requirements that arise and become part of the system despite having no benefits as per the established procedures. By design, UgCD should ensure all those requirements are captured that help to accomplish each possible task and thereby ensure high software usability. Notably also, by task modelling, the focus is on the actual intended software usage and the software under UgCD is designed by engineering it to make it especially support task accomplishment, which is what should make the software highly usable. The method that will be followed in this study for e-learning software is detailed in the methodology chapter that follows.

Table 11: Simple example of user task and its system requirements

User task	System requirements
Identify myself	• Request identity of user • Verify identity of user

5.6.3 Contrast between requirements gathering under UgCD and UrCD

Section 3.4 pointed out many important differences between UgCD and UrCD that were then summarised in Table 3. As far as requirements gathering is concerned, two key differences are prominent:

1. UgCD involves uses (more heavily and selectively) whereas UrCD is merely user-centred. The UgCD process is therefore much more collaborative.

2. UgCD takes an essential approach on which there is a focus on satisfying actual needs and key requirements.

Moreover, as was highlighted in Table 3, UgCD is a fully specified and systematic process (driven by models) whereas UrCD is typically unspecified and user input is informal or varied. The table below summarises the key differences between UgCD and UrCD with respect to requirements gathering.

Table 12: Contrast between requirements gathering under UgCD and UrCD

	User-centred (UrCD)	**Usage-centred (UgCD)**
Focus & objective	Users in order to improve user experience/satisfaction	Usages/Activities in order to support task accomplishment
Nature and extent of user involvement	User centric but informal/varied involvement	Selective and collaborative in a fully specified and systematic way
Design procedure	Iterative prototyping by trial and error	Abstract modelling and refinement
Specification	User descriptions/characteristics	User-system relationships

Chapter Six

Research Methodology

Chapter 6: Research methodology

6.1 Introduction

This chapter reiterates the aim of this research and the research questions framed to achieve this aim from chapter 1. It then details the research design that will be employed to carry out the research, and describes the e-learning software and Saudi higher education institution contexts chosen to demonstrate the requirements recapturing process. More precise details are then provided of the sampling methods that will be used to obtain the participants and of the samples themselves, the measures that will be made, and of the research instruments that will be used. Notes on the ethical considerations are also provided. A draft of the research schedule is provided at the end.

The aim of this research is to show how UgCD can be used to improve the capture of usability requirements and to then develop a framework to guide software developers to do the same for the ultimate objective of improving the usability of the software. The main research question framed to achieve this aim was as follows:

1. What framework can guide developers to better capture usability requirements?

In order to address this research question, the following sub-questions were framed:

2. How are usability requirements being captured at present?

3. How well are existing requirement capturing processes able to capture usability requirements?

4. Does UgCD enable better capture of usability requirements?

The following sections of this chapter detail precisely how the research will be carried out in order to answer these research questions.

6.2 Research design and methods

6.2.1 Research design

This research will employ a complex mixed methods research design that will involve different qualitative and quantitative research methods and steps. This research design is outlined in Figure 20 below and a simplified version is outlined in Figure 21 further below showing only the first two phases with the latter split into two sub-phases. A mixed

methods research design has been chosen because of the possibility of obtaining a better understanding of the situation than would be possible using either one type of method on its own (Creswell & Clark, 2007). Moreover, it was justified because of its powerful potential for obtaining useful, high quality, exhaustive and balanced information (Johnson et al., 2007).

The first phase of the research will involve gathering data pertaining to the present as-is state of how requirements are captured, and on information and awareness concerning usability aspects and knowledge of UgCD. Surveys and interviews will be used during this phase. The second phase will involve recapturing requirements using UgCD and then testing for improved capture afterwards to show how much the requirements are now more complete and precise than before. The choice of the two qualities of completeness and preciseness is based on the IEEE recommendations mentioned in 5.3.3 and Harschnitz (2011)'s identified conditions for requirements to be deemed effective mentioned in 5.4.2.

Although the research will be considered as complete at this stage, developers will be provided with sufficient recommendations in the form of a framework for implementing UgCD for capturing requirements that they will be able to use to easily implement UgCD and hopefully achieve improved usability. That is, participant developers may become convinced of the possible superiority of the improved requirements capturing process and desire to continue with the design and to implement it. But, any such implementation of improved software designs to test for usability improvements are not likely to be done due to time limitations, as this is not a longitudinal study.

Figure 20: Model of research design

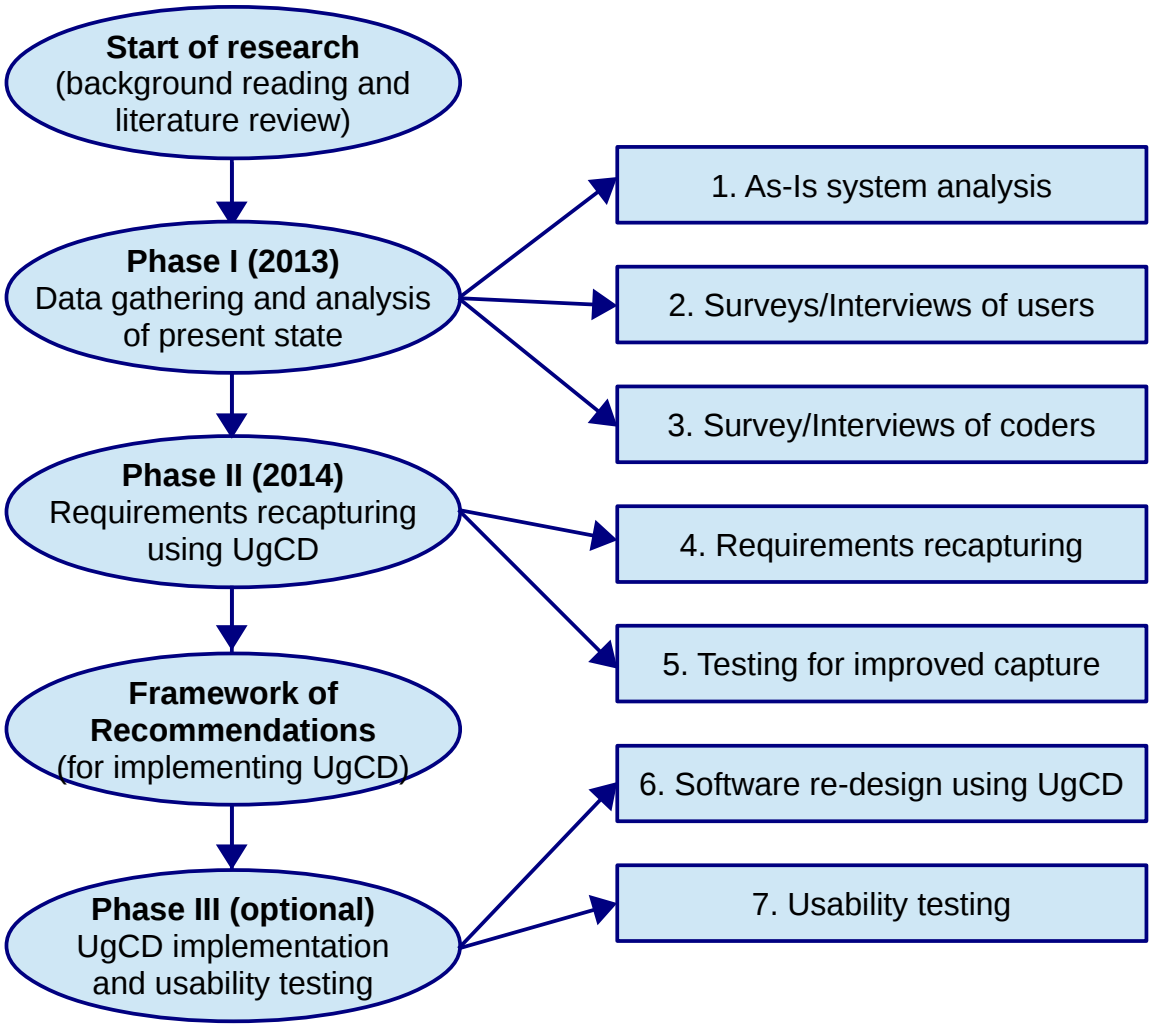
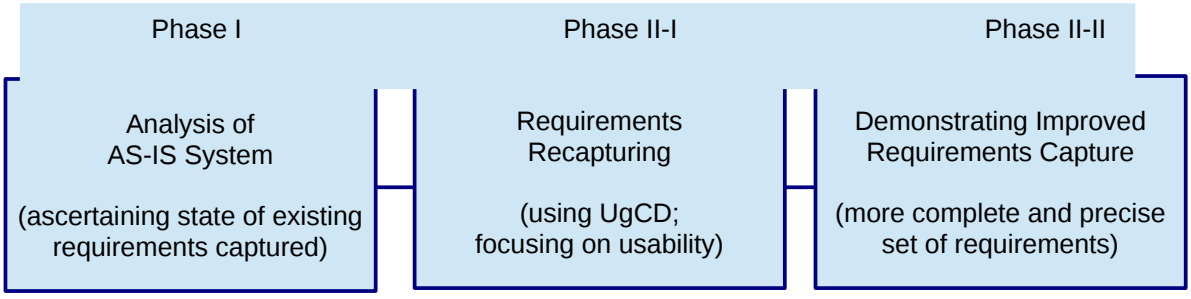


Figure 21: Simplified model of research design



The study will first ascertain how well existing software in use is able to capture the software requirements for its users. After adjusting the software development phase using UgCD instead, the same participants will be asked again if the new approach is better able to capture their requirements in the second phase. The software in its present state may be

unreliable, expensive to use, lacking in functionality, insecure, etc. so the specific aim is to address the usability aspect to ultimately make it possible to adapt and improve the quality of the software. It is anticipated that improvement will be seen in being able to better capture software requirements for improved usability in terms of completeness and preciseness. The change will be in the way UgCD through its focus on usage more so than on the user, can improve the capturing of software requirements, specifically meeting usage requirements. That is, the concern will be with satisfying the usage requirements for which the software is designed in the first place. The strongest elements of both agile and plan-driven approaches will be combined and the overall design will be usage centred. The reason for this is so as to ensure better coverage of requirements.

6.2.2 Research methods

The decision to use a mixed methods design during the first (data gathering phase) of this research is justified on the basis of it being able to provide a better understanding of the current situation than would be the case if only one of the two (i.e. qualitative and quantitative) approaches was used (Creswell & Clark, 2007). The methods that will be adopted are surveys for the quantitative part and interviews for the qualitative part. Their role and justification for use are described briefly below, and the content of the instruments and how they will address the research questions are detailed in section 6.5.

The surveys will enable data to be obtained from a large sample of e-learning software users (teachers and students) and software developers whereas the interviews will be focused on a select number of individual developers for more in-depth insight into how requirements are captured at present. The survey method is considered appropriate because it allows large amounts of data to be gathered from a sizeable population in a highly economical manner (Saunders et al., 2009: 110). The three survey questionnaires will be made available at http://kwiksurveys.com for the convenience of all participants to complete in their own time within a one-month period.

The interviews will be conducted in the meantime so the two parts of phase I will be conducted concurrently rather than sequentially. The interview method is useful because it allows information to be investigated more deeply than is possible using a quantitative method (Saunders et al., 2009). Moreover, interviews allow for guided conversations to be held (Yin, 2003), and the deeper insight could help to open up new dimensions into real life experiences (Saunders et al., 2009). As far as this study is concerned, the insight will

focus on how requirements are being captured at present as well as on knowledge and awareness of usability and UgCD.

The second phase of this research will involve a practical demonstration in which UgCD will be implemented for capturing usability requirements in order to show how such requirements can be 'better captured'. The second sub-phase of phase II will then seek to establish that UgCD has indeed captured usability requirements in a more complete and precise manner.

6.3 Selected software for demonstrating improved requirements capture

The framework of recommendations for developers, which will detail how the UgCD process can be used for capturing requirements, will be applicable to software in general. However, e-learning software will be used merely for the purpose of demonstrating the possibility of improved requirements capture using UgCD prior to devising the framework. Specifically, e-learning software will be chosen that is in use at institutes of higher education in Saudi Arabia, and whose main users are teachers and students. The e-learning systems will have already been developed so the underlying aim will be to improve the usability of these existing and already implemented systems. This contrasts with typical studies which involve creating software from scratch. As such, the requirements analysis technique that will be used will be geared to re-engineering, i.e. a new list of of software requirements will be prepared that could then be used, at the discretion of the e-learning software programmers, for an attempt at software re-engineering that could in turn lead to an improvement in the usability of the software. The study will therefore focus exclusively on the software design and development phase, and will not involve assessing the potential benefits to teachers and students in terms of teaching and learning through using the software. After checking for improved usability however, the institutions may decide to implement the improved software and to then test for possible improvements in students' academic performance.

The reason for selecting e-learning software is that they typically have a lot of users among a fairly homogeneous group, i.e. students and teachers. This makes this context ideal for administering a large-scale survey. Also, importantly, these users are readily accessible as potential respondents because they work or study at the same institution. For other general types of software, it may not have been possible to have access to users in a similar convenient way, let alone gain access to a large number of potential respondents. In short,

it is anticipated that the selection of e-learning software will help to address the research questions quickly, conveniently, and with a sizeable sample, and thereby make a useful contribution on requirements capture and usability.

Moreover, by selecting already created software, its existing usability can be checked easily by asking its users who will already have been using the software for quite some time and will be familiar with it. If software had to be created from scratch, it would take time not only to develop it, but also for users to become familiar with it and be able to comment on its usability. That is, it is easier for users to describe how usable an existing software is, which they have been using, than it would be to ask them to try a new software, and that too after developing it first. This should help to get results more quickly.

6.4 Sampling

The sample of coders and e-learning software users will be obtained from across 5 universities located in three cities in Saudi Arabia. Two of these will be from Jeddah, one of which has been preselected to be King Abdulaziz University wherein the earlier research on agile methods was conducted. A further two will be selected at random from Riyadh and one from the city of Taif. This information is stated in Table 13 below. This research will attempt to capture requirements for only two categories of users of e-learning software, namely teachers and students. It will not therefore take into account the roles of course administrators or other users.

Table 13: Universities at which the investigation will be conducted

City	Number of universities that will be investigated
Jeddah	2 (including KAU)
Riyadh	2
Taif	1

After the five institutions have been selected, an estimate will be made of the total number of students who use e-learning software across all of them. This figure will then be used to derive an appropriate size for the student survey sample, either using Jabawi software, or alternatively an online sample size calculator at http://www.raosoft.com/samplesize.html. The same will be done for the other samples.

Assuming the total student body exceeds 200,000, for a 95% level of confidence and 5% margin of error, the minimum recommended sample size is 384, derived from the online sample size calculator. It is anticipated that a sample size of the order of at least a few hundred (about 400 to 500) will be obtained easily for the sample of students who use e-learning software given that the number of students would number several tens of thousands in each university. A target for the sample of teachers who use e-learning software has been set at a minimum of 50, as an attempt will be made to obtain at least 10 teachers (or faculty members) from each university. The sample of software developers is expected to be around 15-20 assuming at least 3-4 developers will accept the request for participation in the survey from each institution. This latter survey sample of developers will then be used from which to draw the sample of developers for interviews, which are expected to be only a few in number given the time consuming nature of the interview method. An attempt will be made to conduct one or two interviews at each of the five institutions which would yield a total number of 5 to 10 interviews. It will be considered sufficient to obtain findings from only a few persons directly responsible for requirements gathering at each institution to avoid the task of making intra-institutional comparisons and sorting duplicate data.

Table 14: Summary of sample parameters for phase I of the research

Sample	Sample 1	Sample 2	Sample 3	Sample 4
Population	University students... who use e-learning software at the five universities	Teachers or faculty members... who use e-learning software at the five universities	Software developers who develop e-learning software at the five universities	
Method	Survey			Interviews
Expected Size	400-500 (approx.)	50 (min.)	15-20	5-10
Size per institution	80-100 (approx.)	10 (mn.)	3-4	1-2

6.5 Research instruments

6.5.1 Overview

This research will involve the use of two types of research instruments for gathering data on existing requirements capturing processes in terms of how well they provide for software usability and how the improved processes will be able to achieve the same. These two instruments are a set of interview questions that will be targeted at e-learning software developers, and a set of survey questionnaires that will be targeted at each of the two main types of users, namely teachers (or faculty members) and students, and the software developers directly responsible for gathering software requirements.

The table below summarises the information about the research instruments, the types of information they are designed to gather, and links them to the specific research questions that they will target. The first two samples that will be surveyed, i.e. comprising of *users* of e-learning software, will be asked about various aspects of usability based on their experiences, and the aim will be to target RQ3 to find out how well the existing requirements capturing processes are able to capture usability requirements. The last two samples for which both (survey and interview) methods will be used, which will comprise of e-learning software developers, will be asked a wider range of questions pertaining to software usability, requirements capturing and UgCD as indicated in the table and detailed in section 6.5.3.

Table 15: Research instruments and targeted research questions

Samples	Samples 1 and 2	Samples 3 and 4
Population	USERS of e-learning software at the five universities (students and teachers or faculty members)	DEVELOPERS of e-learning software at the five universities
Research instrument(s)	Survey questionnaires	Survey questionnaire and interview questions
Type of information to be obtained	• Software usability experience	• Software usability awareness • Usability requirements • Usability requirements capturing • How usability is ensured • Usability testing • Awareness of UgCD
Main targeted research question	3 (How well existing requirement capturing processes are able to capture usability requirements)	2 (How usability requirements are being captured at present)

6.5.2 Surveys of software users

The survey questionnaires devised for the students and teachers are contained in Appendix D and Appendix E respectively.

6.5.3 Survey and interviews of developers

The survey and interview questions devised for developers are contained in appendices F and G respectively. This survey questionnaire consists of twelve questions of which the first ten constitute a list of usability aspects for which a six-point Likert scale has been devised. A Likert scale is useful because it is relatively easy and quick to construct, it is reliable, each statement in the scale is given an empirical test for discriminating, and it can be used in either respondent or stimulus-centred studies (Kothari, 2004: 86). The choice of a six-point scale was made with a view to maximising the possible reliability of the scores on one hand while avoiding too many response categories on the other that would make it difficult for respondents to discriminate between them. The increased reliability of larger point scales is well supported, for example by Preston & Colman (2000) although they actually recommended 7 to 10 response categories for even greater reliability.

The purpose of the questionnaire is to ascertain the extent to which the developers consider the software to be capable of capturing requirements to ensure each of the usability aspects. The aspects are worded indirectly and using simple language so as to make it easy for respondents to complete the questionnaire. The key usability aspect to which each of them pertains is listed in the table below. The other two questions ask the respondents to identify the methods used to capture requirements and the methods used, if any, to check for usability afterwards. The interview questions consist of 11 items, which are targeted at various aspects related to capturing software usability requirements, understanding usability, ensuring usability, testing for usability, and awareness of UgCD.

Table 16: Developer survey items matched with usability aspects

Number	Wording used	Main usability aspect
1	Minimal learning time to use the software	Learnability
2	Ease of use of the software	Learnability
3	Following of established standards in design	Efficiency
4	Incorporation of metaphors from people's real world experiences	Learnability
5	Facilitation of users in successfully accomplishing intended tasks	Efficiency
6	Minimisation of possible errors	Reliability
7	Consistently enabling quick completion of tasks	Efficiency
8	Reliability of the software to work as expected	Reliability
9	Aesthetic appeal of the software	Satisfaction
10	Ability to engage the user	Engaging

6.5.4 Reliability and validity of the survey questionnaires

All three survey questionnaires are original instruments specially prepared for this research so it will be necessary to establish their reliability and validity. Reliability is assured by checking whether the results a questionnaire provides are consistent, accurate and replicable. Validity is assured by checking whether the results actually measure what is intended to be measured. Establishing both of these is important because they determine the quality of the research instruments (Patton, 2002).

The research instruments devised will be checked for their reliability and validity by requesting the researcher's supervisor to check through the choice of questions and the appropriateness of the way they have been framed. The use of peer assessment will also be relied upon for this purpose, as recommended by Easterby-Smith (1991). In addition, a small scale pilot research will be conducted at the researcher's own institution (Southampton University) among a select number of students, teachers and software developers. It is also pertinent to note that the real study will be conducted at five different universities in Saudi Arabia, so this also will add to the validity of the research results and findings.

6.6 Data analysis

Data gathered from the three survey samples of e-learning software users (faculty members and students) and developers will be analysed by providing descriptive statistics of the information relating to usability, requirements capturing and UgCD. Statistical analysis software will be used to assist in the data analysis due to the sheer number of expected respondents and responses. For this purpose, PSPP software will be used, which is an open-source and easily available equivalent of SPSS, and NVivo may also be used if appropriate.

As mentioned in section 6.5.1, the first two surveys targeted at users will gather data pertaining to their usability experience and will help answer the third research question, and the analysis of the data gathered from the third and fourth samples comprising of software developers, will provide information to answer the second research question. The qualitative information from the five interviews will be analysed by condensing, comparing and coding. Key points will be gathered from the responses from each institution so as to extract and condense the useful information, they will be compared with each other in order to identify any useful parallels or differences, and finally, coding will assist to categorise the information.

Although the third research phase identified may not be carried out, it will left at the discretion of the software developers to implement the framework to be recommended to help them redesign their e-learning software so that they could gain practical benefit from the research. The ultimate aim is to enable software usability to be improved. Any benefit from the research could then be ascertained by considering the following stages of analysis:

Stage 1: A working (functional/usable) software?

Stage 2: Positive outcome of using the software?

Stage 3: Are end users able to gain/learn from its use?

Stage 4: Evidence of any organisational changes?

If the institution does implement the changes recommended as the outcome of the research, then these changes will be examined in order to compare them with the previous state for evidence of improvement in usability.

6.7 Objective measurement

6.7.1 Quality measures

This study involves measuring the quality of a software, that is, measuring the functional (external) quality of software as opposed to its structural (internal) quality. Specifically, it is about how well the software produced using UgCD is able to capture requirements that impinge upon usability of software used for e-learning. It is understood that measuring quality is a subjective state that is not directly quantifiable and therefore the measurement is not necessarily exact. However, measuring quality subjectively is important because this factor is inherently subjective as quality can be perceived differently by different people. But, the qualitative measures will be supplemented by some objective measures as well. The table below gives examples of possible subjective and objective measures of quality that could be measured.

Table 17: Examples of subjective and objective measures of software quality

Subjective measures of quality	Objective measures of quality
• Visual appeal • Satisfaction in using • Support for communication/interactivity • Ease of use	• Availability of features • Responsiveness/Speed • Security • Information retrieval • Reliability/Efficiency/Stability • Number of errors • Power consumption • Network usage • Space consumed

6.7.2 Adopted measures

Two key measures will be used to determine whether UgCD does indeed enable better capture of requirements during the second phase of the research. These measures will be of completeness and preciseness, which will measure the extent of completion and precision (or accuracy) in the new list of requirements. The literature supports the importance of these two qualities, as mentioned in section 5.4.2. It is to be noted that being more complete is the opposite of being incomplete and being more precise is the opposite of being imprecise or vague. It is expected that the requirements of the as-is system are incomplete and vague with respect to usability aspects, and it is anticipated that the UgCD process for requirements gathering will enable us to obtain a more complete and precise list of usability requirements. This will be achieved through task modelling with the aim of improving the system's usability, and it will be tested using the measures of completeness and preciseness as stated.

Use of these measures for 'better capture' is justified because it is not possible to say, for example, that a software is 98% usable; rather, certain user profiles, set of tasks, context of use, etc. can be specified and then a particular measure can be used, such as user satisfaction, error rate, task 80 completion time, task completion rate, etc. Both of the adopted measures will be made at two key stages. The first will be of the as-is system by obtaining a list of the captured requirements from the developers, and the second will be after the survey when the requirements will be recaptured in a way as suggested by UgCD.

Both lists of requirements will be subdivided into usability and non-usability requirements, and the usability requirements will be further divided according to the five chosen usability aspects, as illustrated below. These usability aspects were selected based on the literature review, but the pertinent details related to their definition, suitability, and indications, are given in the next subsection. This distinction between usability and non-usability requirements will be made to ensure that the assessment of completeness and preciseness will be done only on the usability requirements captured, and the division into the five usability aspects will be made to find out if either of the two methods is better able to capture specific types of usability requirements.

The two attributes for completeness and preciseness were defined as follows:

Completeness = How well the usability requirements are able to be captured

Preciseness = How descriptive (or clearly expressed) the captured requirements are of the actual user requirements

For assessing the first attribute of completeness, the total usability requirements captured under both methods will be divided into the following three categories labelled as R1, R2 and R3:

- R1: Usability requirements captured under UgCD but not captured already
- R2: Usability requirements captured before but not under UgCD
- R3: Usability requirements captured by both methods

It is anticipated that UgCD will enable a considerable number and type of usability requirements to be captured that have not been captured previously under the existing method in use, so the R1 category should therefore comprise of a large number of requirements. This will therefore be the key determining category because the larger the requirements listed under R1, the greater will be the likelihood of UgCD enabling 'better' capture of usability requirements with respect to completeness. Also, given the greater focus of UgCD on usability, it is not expected for UgCD to miss the capture of any usability requirements that have already been captured previously. If there are any though, they will come under the R2 category. Many of the usability requirements captured under UgCD however, should already have been captured. These requirements which have been captured by both will come under the R3 category.

It is understood that some requirements can be broken down into smaller requirements, so it is possible that under one method the 'whole' requirement is captured whereas the other method captures the same requirement as segments, either fully or partially. This will be kept in mind and noted where possible instead of making a quick quantitative comparison in the form of a simple count of the number of captured requirements under each of the three categories. After having accounted for this possibility, and identifying the different areas or aspects of the software for which usability requirements will have been captured and recaptured, it can then be said that the method yielding a more complete list of usability requirements will be the one with the longer list in each of the five categories, and also in the sum total of the usability requirements, obtained as follows:

Total number of usability requirements = Number of learnability requirements + Number of rememberability requirements + Number of efficiency requirements +

Number of reliability requirements + Number of user satisfaction requirements

The second attribute of preciseness will be assessed with assistance from a group of experts who will be asked to review the wording used in both sets of captured requirements closely and from a comparative perspective. They will be tasked to rate each set of requirements pertaining to distinct areas or aspects of the software identified during the assessment for completeness according to how clear or vague the description is on a four point scale, as follows:

1. The description of the requirement is very explicit.

2. The description of the requirement is reasonably clear.

3. The description of the requirement is not so clear.

4. The description of the requirement is too vague.

The four point scale ranges from very explicit (1), the highest rating, to too vague (4), the lowest rating. Requirements given a high rating (1 or 2) will be considered as being precise statements whereas those given a low rating (3 or 4) will be considered as being imprecise statements. The greater the number of high ratings in relation to the total number of usability requirements for each method, the greater will be its assessment of preciseness.

Both measures will be made for the requirements already gathered for the as-is system and for the requirements that will be recaptured using the procedure suggested in UgCD. If the 'values' for the already captured requirements under the present software development method are specified as C1 and P1 for completeness and preciseness respectively, and the corresponding values for UgCD as C2 and P2, then the implication of UgCD enabling better capture of usability requirements will be made if the following conditions are satisfied:

C2 > C1, i.e. if a more complete or longer list of usability requirements are captured under UgCD (pertaining to distinct areas or aspects of the software)

P2 < P1, i.e. if a larger proportion of the captured usability requirements are given higher preciseness ratings under UgCD

It is anticipated that under UgCD, the values obtained for the variables C2 will be much greater and for P2 will be much less than those obtained for C1 and P1 due to the greater focus of UgCD on usability. It is to be noted that a higher value of C indicates a greater

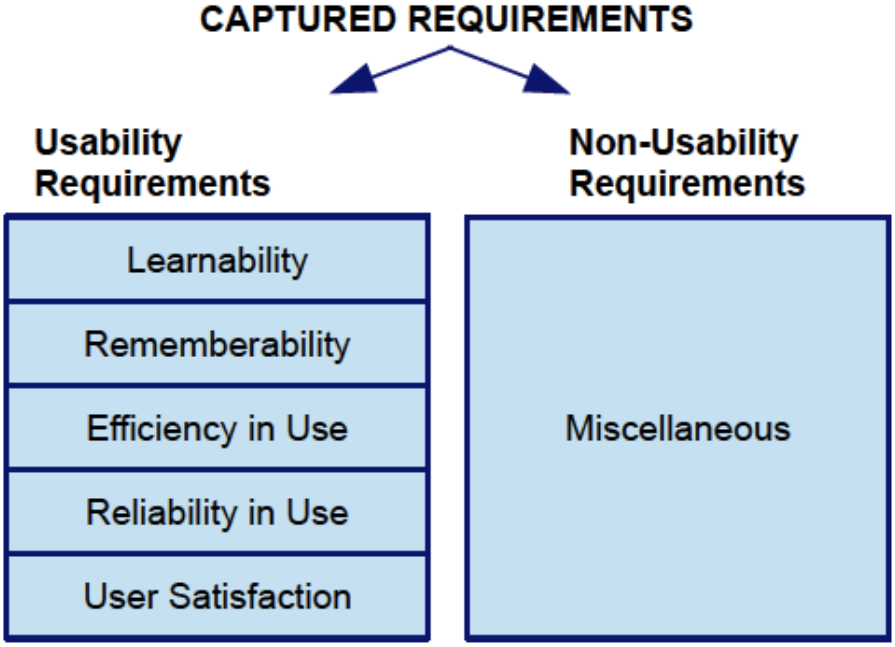
extent of completeness, and a lower value of P indicates a greater degree of preciseness, the latter by being closer to the value of 1 (out of 4) specified for the most precise rating.

6.8 Justification for the selected usability aspects

6.8.1 Selected usability aspects restated

The selection of the five usability aspects are learnability, rememberability, efficiency, reliability, and satisfaction (Figure 22). These are justified based on their suitability for the context of e-learning, as identified by Constantine (2011) and pointed out in 4.7.1 Usability aspects for e-learning software. Apart from rememberability, these usability aspects are also recognised as classical attributes, which were examined in 4.7 Software design for usability in comparison to other conceptualisations of usability. In particular, as pointed out by Hammond et al., (2002), usability requirements are most closely associated with being able to enhance a software's efficiency and satisfaction. All five of the selected usability aspects are detailed in turn with respect to their definition, suitability and indications. The relevant details have been extracted from the literature review.

Figure 22: Division of captured requirements



For the purpose of detailing the definitions, suitability and indications of the five selected

usability aspects, these are categorised as pertaining to either cognitive, technical (or performance) or affective aspects of the user experience, as described therein. Also, besides the identification of the selected five usability aspects by Constantine (2011), for their indications, two additional supporting sources were found, namely Lausen & Younessi (1988) and Bachman (2004). These serve to show the commonality of the selected software quality aspects as pertaining to usability, which were then reiterated by Constantine. Lauesen & Younessi identified four of the aspects (with the exception of reliability) as usability factors in their study. The terms they used were 'ease of learning', 'task efficiency', 'ease of remembering', and 'subjective satisfaction' to which they added 'understandability'. Bachman defined all five aspects referring to them as learnability, efficiency, error tolerance (for reliability), memorability (for rememberability), and satisfaction (for user satisfaction).

Table 18 presents a summary of the aforementioned two sets of indications for the five selected usability aspects, and the final column relates these to the items in the survey questionnaire that were matched with these usability aspects earlier in Table 16. The first column in the table lists key supporting definitions or studies from the literature review. The three aforementioned studies support all five selected attributes (except one which mentions only four): (1) Lausen & Younessi (1988), (2) Bachman (2004), and (3) Constantine (2011). The other studies listed support one or more, but not all, of the selected attributes.

Table 18: Indications for the five selected usability aspects

Key supporting definitions/studies	Question presented by Bachman (2004)	Criteria specified by Lauesen & Younessi (1988)	Items in the survey questionnaire devised
LEARNABILITY (cognitive attribute)			
-IEEE Standard 1061 (1992) -ISO 9126 -Lausen & Younessi (1988) -Nielsen (1994) -Bevan & Azuma (1997) -Kobbenes & Folkman (2003) -Venkatesh et al. (2003) -Bachman (2004) -Manzo (2010) -Constantine (2011) -Dubey et al. (2012)	"How quickly do users come up to speed on the product?"	"The system must be easy to learn for both novices and users with experience from similar systems."	-Minimal learning time to use the software -Ease of use of the software -Incorporation of metaphors from people's real world experiences
REMEMBERABILITY (cognitive attribute)			
-Lausen & Younessi (1988) -Bachman (2004) -Constantine (2011) -Dubey et al. (2012)	"Do users remember how to use the product between uses?"	"The system must be easy to remember for the casual user."	(User survey)
RELIABILITY (technical attribute)			
-Bachman (2004) -Constantine (2011)	"Do users make few errors? Are [those] errors recoverable?"	(Not specified)	-Minimisation of possible errors -Reliability of the software to work as expected
EFFICIENCY (technical attribute)			
-ISO/DIS 9241-11 -Lausen & Younessi (1988) -Hammond et al., (2002) -Bachman (2004) -Constantine (2011)	"How easy is the product to use and be productive?"	"The system must be efficient for the frequent users."	-Following of established standards in design -Facilitation of users in successfully accomplishing intended tasks -Consistently enabling quick completion of tasks
SATISFACTION (affective attribute)			
-ISO/DIS 9241-11 -Lausen & Younessi (1988) -Bevan & Azuma (1997) -Feldstein (2002) -Hammond et al., (2002) -Bachman (2004) -Rubin & Chisnell (2008: 4) -Constantine (2011) -Dubey et al. (2012)	"Do users enjoy using the product?"	"The user must feel satisfied with the system."	-Aesthetic appeal of the software -Ability to engage the user

6.8.2 Learnability and Rememberability

Learnability and rememberability are considered together, as they both pertain to cognitive

aspects of the user experience. For e-learning software, the learnability aspect especially is one of the most important measures of usability (Kobbenes & Folkman, 2003) because it leads to supporting the extent of being able to learn the content. Learnability in general, for any type of software, refers to the capability of the software itself to be easy to learn to use that makes the user able to use it to get work done quickly. It is therefore determined by the time spent in learning how to use the software (4.3 The importance of usability). Nielsen (1994) referred to this aspect as ensuring the software is 'easy to learn' (Table 6 and Figure 8). Manzo (2010) described it together with 'practicability' as relating to conceptual requirements, as distinct from more organisational requirements, and also from efficiency as a more functional requirement (Table 7). Dubey et al. (2012) defined both learnability and rememberability together as a single 'comprehensibility' attribute, as ““The degree to which the software has clarity, is easy to learn and remember and includes appropriate help/documentation.” It should be noted that learnability and rememberability refer to qualities of the software being developed, and not whether the material being taught is learnable or rememberable.

Rememberability alone refers to enabling the user to easily remember the system so that it can be used easily again after a lapse of time. It is therefore measured by asking users how well they “remember how to use the product between uses” (Bachman, 2004). Specifically, the question Bachman posed for learnability was “How quickly do users come up to speed on the product?”, and for ease of remembering, “Do users remember how to use the product between uses?” Lauesen & Younessi (1988) specified the criteria for ease of learning as, “The system must be easy to learn for both novices and users with experience from similar systems.”, and for ease of remembering as, “The system must be easy to remember for the casual user.” In the survey instrument, learnability is indicated by minimal learning time, ease of use, and incorporation of metaphors from people's real world experiences, and rememberability by factors that may be carried over between uses.

6.8.3 Reliability and Efficiency

Reliability and efficiency can be said to relate to the technical performance of the software. As such, these are often identified as distinct from usability as well as functionality, as in the standard ISO/IEC 9126-1. Similarly, Harschnitz (2011) regarded this as a technical requirement as opposed to a functional requirement (5.2.2 Functional requirements). However, these two aspects also affect user perceptions of usability, hence their inclusion

as usability aspects by Constantine (2011), and the way they are regarded in this study. Dubey et al. (2012) defined an attribute of 'effectiveness', which can be related with reliability and efficiency, as “The degree to which the software facilitates the user in accomplishing the task for which it is intended with precision and completeness while avoiding most errors in varying contexts of use.”

In the survey instrument, reliability was measured by the capability of the software to facilitate users in accomplishing tasks and for the software to work as expected without encountering any errors. Efficiency was indicated by the same, the following of established standards, and being able to consistently complete tasks quickly. Efficiency therefore enables a high degree of productivity to be gained from using the software efficiently. For efficiency, Bachman (2004) posed the question, “How easy is the product to use and be productive?”, and for error tolerance, “Do users make few errors? Are [those] errors recoverable?” Lauesen & Younessi (1988) specified the criteria for efficiency as, “The system must be efficient for the frequent users.”, and did not specify one for reliability.

6.8.4 User satisfaction

User satisfaction can be singled out from the other four selected usability aspects, as it pertains to more affective aspects of the user experience. That is, it is more subjective in nature, and is determined mainly by the aesthetic qualities of the software and other qualities that make it appealing and attractive. User satisfaction is commonly associated with usability, as pointed out by Feldstein (2002). For instance, it is specifically mentioned as a usability aspect in ISO/DIS 9241-11, and as pointed out in 4.2 Introduction to usability, it is such a quality that its absence in a software would be noticeable and would cause dissatisfaction (Rubin & Chisnell, 2008). In Nielsen's (1994) model depicted in Figure 8, it is a quality that makes a software 'subjectively pleasing', and this subjective association is also highlighted in Table 17.

Dubey et al. (2012) defined satisfaction as ““The degree to which the software is likeable, comfortable, attractive and trustworthy for the users.” In the survey instrument, evidence for this satisfaction aspect was indicated by the aesthetic appeal of the software and its ability to engage the user. Bachman (2004) specified the question, “Do users enjoy using the product?”, as an indication of satisfaction, and Lauesen & Younessi (1988) specified the criteria for 'subjective satisfaction' as, “The user must feel satisfied with the system.” Satisfaction also derives from the support available to users in case of need, such as

through the use of dialogs, and from its ability to engage or stimulate the user, for instance, through interaction, its utility, etc. A questionnaire used in a study by Preece (2001) also related satisfaction with the informational aspects of design, ability to navigate, and responsiveness of the software.

6.9 Ethical considerations

Before administering the survey questionnaires to the three category of participants, namely students, teachers and developers, and conducting the developer interviews, all participants will given all the relevant information about this intention of this reasearch so that they can choose whether or not to participate. Giving this information is necessary so that participants are clearly aware of why the research is being carried out and to alleviate any concerns they might be holding about its purpose (Cozby, 2003: 39). In addition, all participants will be assured of their anonymity and will be told that the results will be aggregated so no individual participant's responses will be identified.

Along with the survey questionnaires and interview questions, two more documents related to the Data Protection Act and ethical issues had to be made available to the users. The first document, called Best Practice (see Appendix F: Best Practice Document), is about the Data Protection Act which relates to subject data that requires it to be “accurate, relevant, not excessive, held securely, retained for only as long as necessary, used only for the purposes of the study, and accessible to the participant”. The second document, called the consent document (see Appendix G: Consent Information Document), contains information to inform the participant of “the School Ethics Committee reference number (E/10/08/003), (see Appendix H: Ethics Committee Application), inform the participant of their right to unconditionally withdraw at any time and for any reason, and (if applicable) shall explain their rights to retain any inducement following withdrawal”.

6.10 Research schedule

The original and later revised research schedules are illustrated in Gantt charts attached in Appendix C: Research schedule.

6.11 Research output

Information will first be gathered about the current (as-is) state of the system in terms of the already gathered or captured requirements as well as the experience of teachers and

students in order to ascertain the extent of usability of the present system. Following this, roles and tasks will be identified and modelled according to the UgCD approach in an attempt to devise a more complete and precise list of usability requirements and show how UgCD is better able to capture requirements. The output of the research will be an improved list of e-learning software usability requirements – improved in the sense of being more complete and precise for achieving greater usability. Assuming UgCD does indeed provide better capture of usability requirements, recommendations will then be made to guide future developers on how they could also implement UgCD to better capture usability requirements. These will address the following stages of the requirements analysis process outlined in Table 19 below.

Table 19: Framework of recommendations of the study

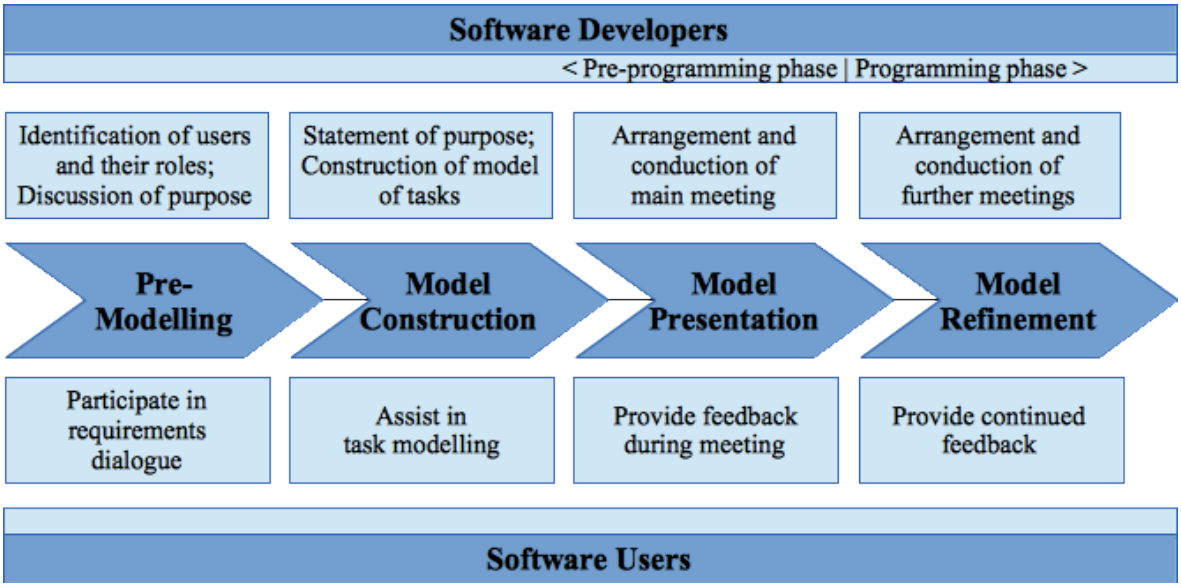
Stage	Details
1. Role/Task identification	Identifying roles and tasks of users through defining essential use cases
2. Task modelling	Modelling tasks with a focus on satisfying software usability
3. Requirements capturing	Capturing usability requirements
4. Requirements testing	Testing for completeness and preciseness of captured usability requirements
5. Requirements refinement	Holding of frequent meetings and continuous consultations to refine the requirements

Although the research will be restricted to the first two phases identified, the third phase is described so as to suggest the possibility for continuation of the project through to implementation of the improved requirements capturing and thereby derived benefits in the form of improved software usability. In the case of e-learning software, this could manifest in the form of improved learning of students, but investigating whether this is the case will be beyond the scope of the study. Participant developers however, may become convinced of the possible superiority of the improved requirements capturing process and desire to continue with the design and to implement it.

Also, it should be pointed out that although e-learning software will be used for the demonstration of better capture and potential enhanced usability, the framework of principles will be devised so as to be generally applicable to all kinds of software. The

structure of the framework will be somewhat like the illustration of the tentative framework shown below. The principles will guide specifically on the role of both sides, which for developers will be mainly related to arranging and conducting while for users it will be mainly related to how best to collaborate with the developers in capturing the usability requirements.

Figure 23: Tentative framework of requirements gathering process under UgCD



6.12 Summary of chapter

This study adopts a mixed-methods research design involving gathering data pertaining to how requirements are captured at present through a survey and interviews with developers, and a phase for demonstrating UgCD for requirements capture. The developers are from higher educational institutions engaged in the development of e-learning software. The survey sample comprised of developers from throughout the Saudi kingdom, the interviews were conducted at institutions located in Jeddah, Riyadh and Taif, and the UgCD demonstration was held at KAU. The instruments devised for the survey and interviews included questions related to all five of the selected usability aspects, namely learnability, rememberability, efficiency, reliability and user satisfaction. Data analysis was conducted using PSPP, and a database to facilitate a thematic analysis of the interviewee responses. The objective measures for the UgCD demonstration in capturing requirements were

completeness and preciseness. The ethical considerations observed and the research schedule followed are also outlined, and the procedure adopted for the UgCD demonstration is detailed.

Chapter Seven

Research Phases and Samples

Chapter 7: Research phases and samples

7.1 Introduction

This chapter briefly outlines the research phases conducted, the methods involved, and the actual research samples obtained as well as the main sample characteristics and demographics.

7.2 Research phases and samples

The actual study involved five stages, which can be grouped into three phases: (1) Pre-UgCD trial survey and interviews, (2) The UgCD trial itself, and (3) Post-UgCD trial survey and interview. This process is depicted in Figure 24. The purpose during the first phase was to gather the views and information about the practices of Saudi software developers working in higher education institutions on aspects related to UgCD. The purpose of the second phase was to demonstrate UgCD by applying it for recapturing requirements to see what difference it can make for software usability, and the purpose of the third phase was to gather the views of the participants at the trial on the UgCD implementation.

Figure 24: The three phases of the research conducted

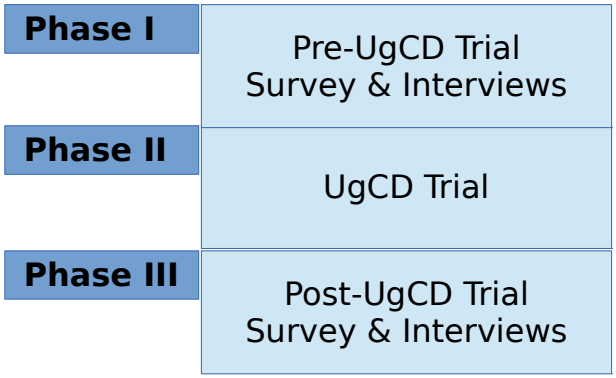


Table 20 below summarises, and Figure 25 illustrates, the samples that were actually obtained in this study during the three phases of the primary research. The main data and analysis for the survey, interviews, trial, and post-trial surveys and interviews are presented

separately over the course of the next four chapters. The pre-UgCD survey data and analysis is presented in Chapter 8: Survey data and analysis, the pre-UgCD interview data and analysis in Chapter 9: Interview data and analysis, for the UgCD trial in Chapter 10: The UgCD trial, and the post-UgCD survey and interview data and analysis together in Chapter 11: Post-UgCD trial.

The research methods involved and the places at which the research was carried out at each stage are also indicated in the table. During phase I, the first survey data was gathered from higher education institutions throughout the Saudi kingdom, and the interview sample from the institutions in the three cities of Jeddah, Riyadh and Taif. The UgCD trial was held at one of these institutions at which a selected interview participant worked as a developer, and the post-UgCD data during phase III was gathered from the same.

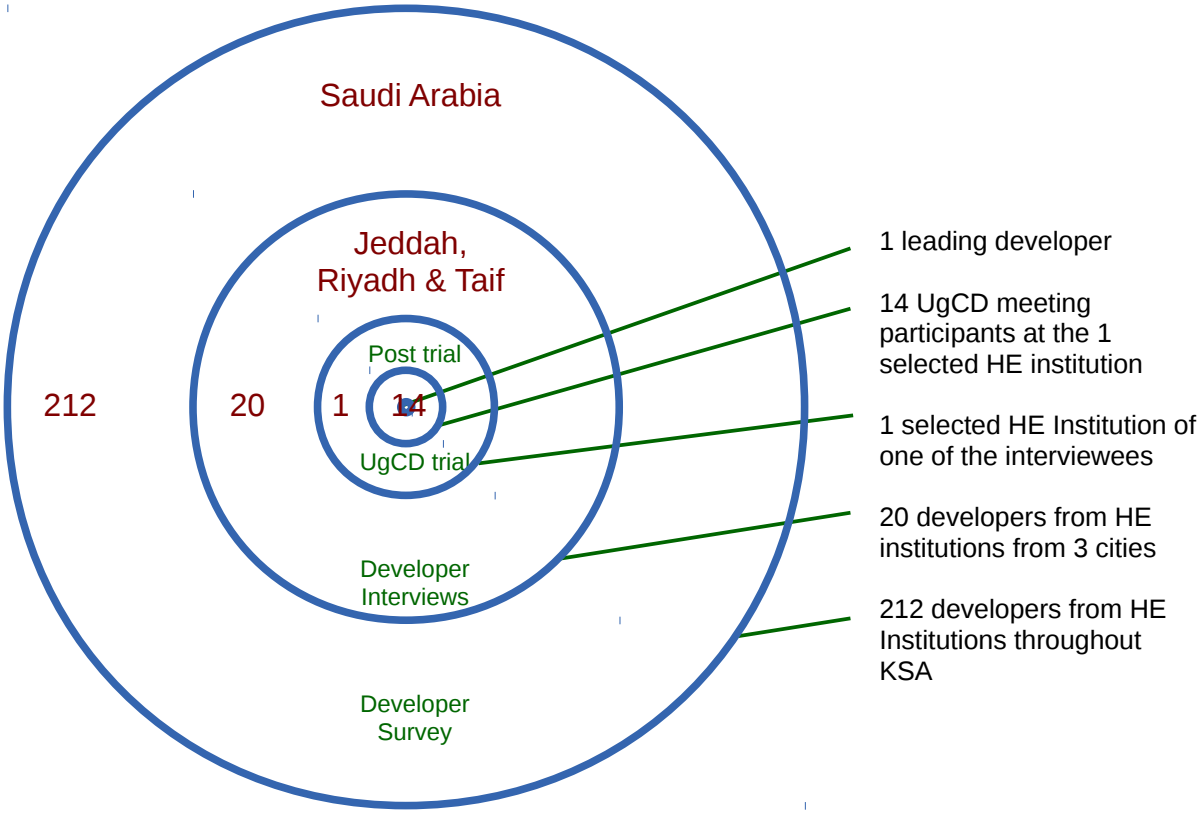
Table 20: Research phases, methods and samples

Phase	Stage	Research method	Location	Sample
I	1	Survey	HE institutions throughout KSA	212 software developers
	2	Interviews	HE institutions in Jeddah, Riyadh and Taif	20 software developers
II	3	UgCD trial	One selected HE institution from above	1 trial
III	4	Post UgCD trial survey	As above (meeting participants)	14 meeting participants
	5	Post UgCD interview	As above (software developer)	1 interview

The final column in the above table mentions the sample sizes at each stage of the research. Only research with developers was permitted as per the ethics applications (Appendix H: Ethics Committee Application), so students were not involved in this study. The number of participants for the pre-UgCD trial survey and interviews were 212 and 20 respectively with the latter being drawn from within the survey sample. A single UgCD trial was then held at the institution of one of the interviewees during which UgCD was demonstrated for gathering requirements. The research during the post-UgCD third phase was carried out among the participants of the meeting that took place during the UgCD

trial. This post-UgCD survey involved 14 participants, of which 4 were developers and the remainder teachers, and the interview was held with one of these who was a developer. Each subsequent sample was therefore a sub-sample of the previous sample. This is made clear in the diagram in Figure 25.

Figure 25: Obtained samples of the study



7.3 Sample characteristics

7.3.1 Survey sample

A total of 212 software developer-respondents from all over the Kingdom of Saudi Arabia completed the online questionnaire[1]. This was a little above the target of 200 developers so the response was good, and this was much higher than the original target of 20 developers, which would have been from only a few selected institutions. It may have been possible to obtain an even larger sample, but the survey had to be terminated at some point. Relative to the target of 200, the response rate was therefore 106%. No demographic data was gathered of the survey sample, but as a criteria for participation, all the participants were

1 This was made available at http://kwiksurveys.com/s.asp?sid=4lqqmrgtm2ws801285657.

software developers appointed in higher education institutions throughout the Kingdom of Saudi Arabia.

7.3.2 Interview sample

A total of 20 interviews were conducted, and all the interviewees were software developers based in higher education institutions in Saudi Arabia from three cities. The institutions were selected on the basis of being in sufficient close proximity for the researcher to be able to visit in person to conduct the interviews wherever the opportunity presented itself although in some of these cases too, interviews were conducted online for convenience by using the Skype chat client. This method of purposive sampling was in contrast to the open random invitations for participation in the survey made throughout the kingdom. Another commonality between the selected institutions was that they were all large universities located in large cities in the western and central regions of the kingdom, namely the three large cities of Jeddah, Riyadh (capital) and Taif. No interviews were therefore held in any of the smaller cities or far flung areas of the kingdom, and the findings can therefore only be considered as representing the views of software developers in universities located in the three aforementioned cities.

7.3.3 Interview dates and venues

Table 21 lists the dates and places of all the interviews and the type of interview, whether conducted face-to-face or online. Interviewee 12 is highlighted because it was at his institution that the UgCD demonstration was carried out.

Table 21: Interview dates and venues

Interview(/ee)	Date of interview	Type of interview	Place of interview
1	12th January 2014	Online	King Abdulaziz University, Jeddah
2	18th February	Face-to-face	Darul Hikmah University, Jeddah
3	19th February	Face-to-face	Darul Hikmah University, Jeddah
4	22nd February	Online	King Saud University, Riyadh
5	24th February	Face-to-face	King Abdulaziz University, Jeddah
6	24th February	Face-to-face	King Abdulaziz University, Jeddah
7	25th February	Face-to-face	Darul Hikmah University, Jeddah
8	27th February	Face-to-face	Taif University, Taif
9	March	Face-to-face	Taif University, Taif

10	March	Face-to-face	Taif University, Taif
11	March	Online	King Saud University, Riyadh
12	March	Face-to-face	King Abdulaziz University, Jeddah
13	March	Face-to-face	King Saud University, Riyadh
14	March	Face-to-face	Arab Open University, Riyadh
15	March	Face-to-face	King Abdulaziz University, Jeddah
16	March	Face-to-face	Darul Hikmah University, Jeddah
17	March	Online	Arab Open University, Riyadh
18	March	Online	Arab Open University, Riyadh
19	March	Online	King Saud University, Riyadh
20	March	Online	Darul Hikmah University, Jeddah

7.3.4 The UgCD trial

As the interview findings show, particularly for question 10, most interviewees were reluctant to try another method to their existing practices. Most in fact were restricted by the control of their management. Only three interviewees stood out as being most inclined to possibly allow UgCD to be trialled at their workplace, namely 3, 9 and 12. Of these, interviewee 12 agreed to discuss the request with his team most promisingly, but there were certain provisos, which somewhat restricted the way in which UgCD was able to be implemented. This selected developer and institution is highlighted in the above table, and the background and context of trial are detailed in 10.2 Background and context.

Only a single trial was conducted to demonstrate UgCD for recapturing requirements for improving an existing piece of software. As detailed in the related chapter, this trial involved guiding the develops on how to apply UgCD for identifying users, roles and tasks, and how to carry out the task of recapturing requirements. These were then done during a single meeting that was attended by the whole team of developers and some teachers representing teaching staff working at the institution as users of the software. The team of developers at the selected institution comprises of five members, three of whom are regular coders, and the other two are a graphics artist and a tester. The total number of participants of the meeting was 16, and the remainder 11 of these were the teachers. For comparison, the requirements gathered during the UgCD trial was compared to the requirements gathered by the developers originally at the time of construction of the software, and secondly to the period before UgCD was applied when requirements were gathered without using UgCD.

7.3.5 Post-UgCD survey and interviews

The short post-UgCD survey and interview during Phase III of the primary research was conducted with 14 of the total 16 meeting participants. These 14 participants included the 5 members of the development team, and the remainder 9 were teachers. All the meeting participants were invited, so the response rate was 88%. The remaining 2 teachers may also have participated, but they were not available because this phase of the research was conducted during the winter holiday season. The final interview was arranged with the leading developer and only one interview was held. This interview was held online using Skype.

7.4 Summary

This chapter introduced the three phases of the research and the five stages involving a survey and interviews prior to the UgCD trial, the trial itself, and post-trial survey and interview. The participants during the pre-UgCD survey and interviews were 212 and 20 respectively, a single trial was held, and the participants during the post-UgCD survey and interview were 14 and 1 respectively. Also, each subsequent sample was a sub-sample of the sample from the previous stage. The next four chapters present the data and analysis pertaining to the primary phases of the research in this study.

Chapter Eight

Survey Data and Analysis

Chapter 8: Survey data and analysis

8.1 Introduction and coding

This chapter presents the results of the survey of software developers in the Saudi kingdom and the findings from the analysis of their responses. The sample characteristics were described earlier in 7.3.1 Survey sample, details of the method of coding and the formula that was applied for calculating the overall weighted values and means are presented in this chapter. The frequency data is presented using tables and charts for each of the three categories of the survey, i.e. capture of usability requirements, and usability capturing and testing methods. The analysis includes ordering each of the three sets of aspects and methods in order to ascertain which of them are most and least common. More thorough analysis is undertaken using factor analysis to derive a possible factor structure, and correlation analysis to establish the significant correlations.

The variables that were assigned codes during the analysis are listed in the table below. There were six degrees of responses in the first part ranging from 'not well at all' as the extreme negative response to 'very well' as the extreme positive response.

Table 22: Codes assigned to the variables for the analysis

Code	Variable	Response
NWA	Not well at all	Extreme negative response
NSW	Not so well	Intermediate responses
U	Unsure	
RW	Reasonably well	
QW	Quite well	
VW	Very well	Extreme positive response
OVW	Overall weighted value*	(Calculated value)
TWV	Total weighted value	

*The Overall weighted value was calculated in order to obtain a single figure to represent the entire set of responses for each survey item taking into account the negative and positive degrees of the responses. Each of the response variables was assigned a weighted value, and in the calculation, all the frequencies for each response were multiplied with their respective weights before being added together to obtain the single representative

value. Both positive and negative values were used so that the overall weighted value reflects the overall response being either affirmative or non-affirmative. Alternatively, a positive scale from 1 to 6 could have been applied as well, but the relative ordering would have resulted the same, and the objective was simply to find out this order. The calculation was made as follows:

OWV = (NWA x -2) + (NSW x -1) + (U x 0) + (RW x 1) + (QW x 2) + (VW x 3)

8.2 Capture of usability requirements

8.2.1 Frequencies and overall values

The first part of the questionnaire asked respondents to indicate how well they thought during the development of the software they designed in the recent past, they were able to capture the requirements for 10 specific usability aspects. A summary of the responses is given in Table 23 and Table 24.

Table 23: Mean, SD and T-Scores of Developer perceptions on ability to capture

	Statement	Mean	St. Deviation	T Score
A1	Minimal learning time	3.57	1.116	50.924
A2	Ease of use	5.74	1.044	57.476
A3	Following established standards	5.71	1.245	47.750
A4	Incorporation of metaphors	4.58	1.094	52.149
A5	Facilitation of users in tasks	4.57	1.069	54.174
A6	Minimisation of errors	4.52	1.224	45.790
A7	Consistently quick task completion	4.47	1.190	45.052
A8	Reliability of working	4.45	1.204	45.582
A9	Aesthetic appeal	4.68	1.170	50.019
A10	Ability to engage user	4.49	1.126	49.424

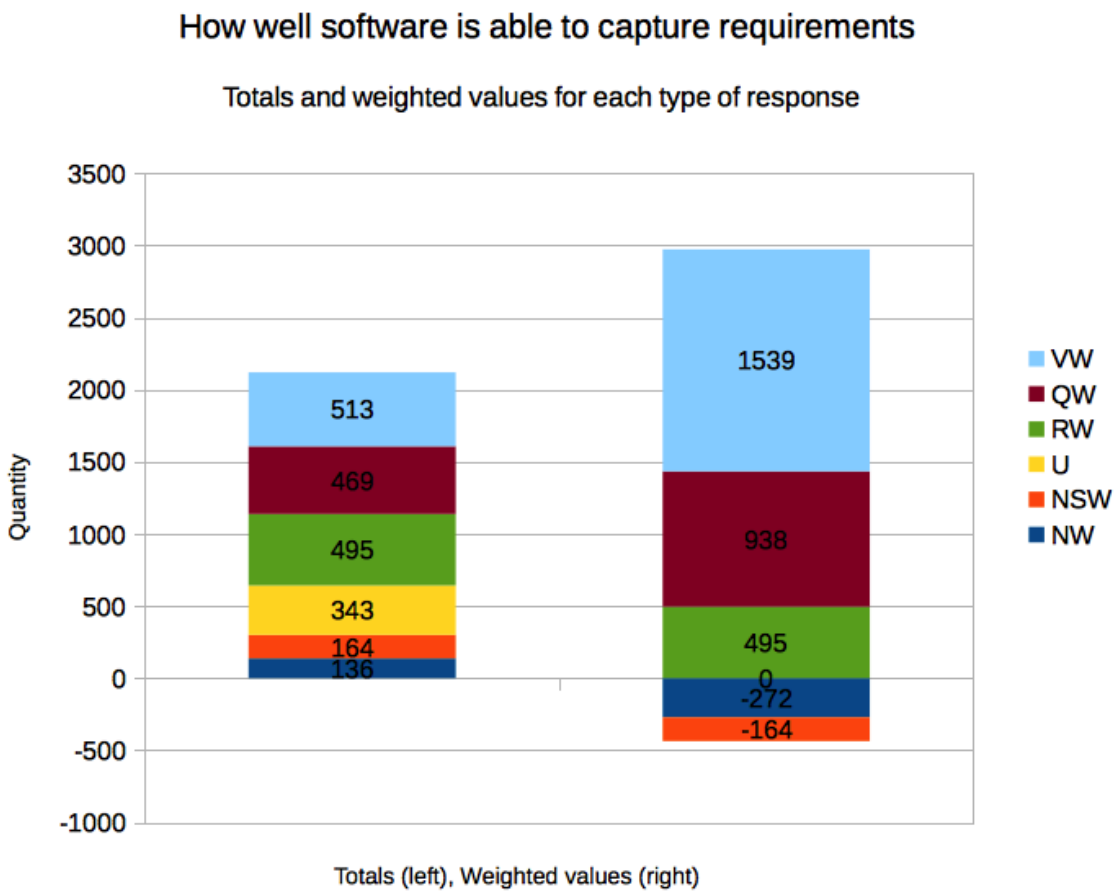
The above table shows the mean, standard deviation and t-scores for each of the ten statements pertaining to the perceptions of the developers on how well they are able to capture requirements for certain usability aspects. The mean values range from 3.57 (for A1) to 5.74 (for A2) so there is a wide variation between them. The pattern of responses is similar in terms of relative magnitude to those in the next table.

Table 24: How well the developers were able to capture requirements for certain usability aspects during recently developed software

VAR	LABEL	NWA	NSW	U	RW	QW	VW	TOTAL	OWV
A1	Minimal learning time	20	13	17	76	54	32	212	227
A2	Ease of use	0	0	21	42	76	73	212	413
A3	Established standards	4	6	5	44	72	81	212	417
A4	Use of metaphors	18	27	39	55	41	32	212	170
A5	Facilitate users in tasks	19	36	49	28	38	42	212	156
A6	Minimisation of errors	10	17	88	29	33	35	212	163
A7	Consistent/Quick task completion	27	18	28	56	39	44	212	194
A8	Reliability	8	6	12	48	51	87	212	389
A9	Aesthetic appeal	11	14	36	76	28	47	212	237
A10	Ability to engage	19	27	48	41	37	40	212	170
	Total	136	164	343	495	469	513		
	Weight	-2	-1	0	1	2	3		
	TWV	-272	-164	0	495	938	1539		

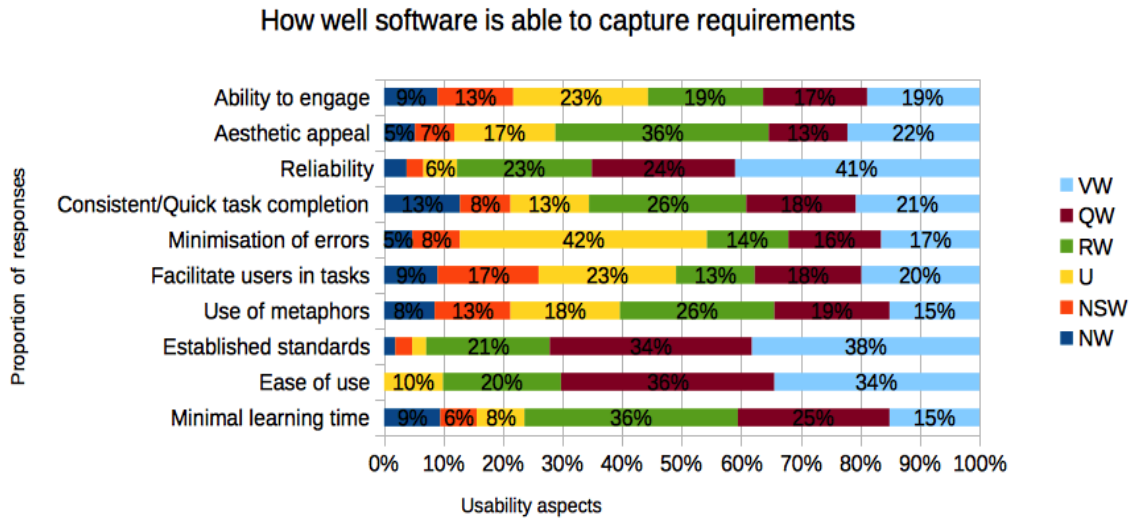
The totals for each degree of response show that the responses are generally skewed towards the higher end of being well captured. This pattern is clearer in Figure 26 below.

Figure 26: Totals and weighted values for each type of response



As can be seen in Figure 27 below, the response of both 'very well' and 'quite well' were highest for A8 (reliability), A3 (established standards), and A2 (ease of use); the greatest uncertainty was for A6 (minimisation of errors) (42%), and at the other end of the scale, the highest proportion of 'not well at all' responses were indicated for A7 (consistent/quick task completion) (13%).

Figure 27: How well the software is able to capture requirements

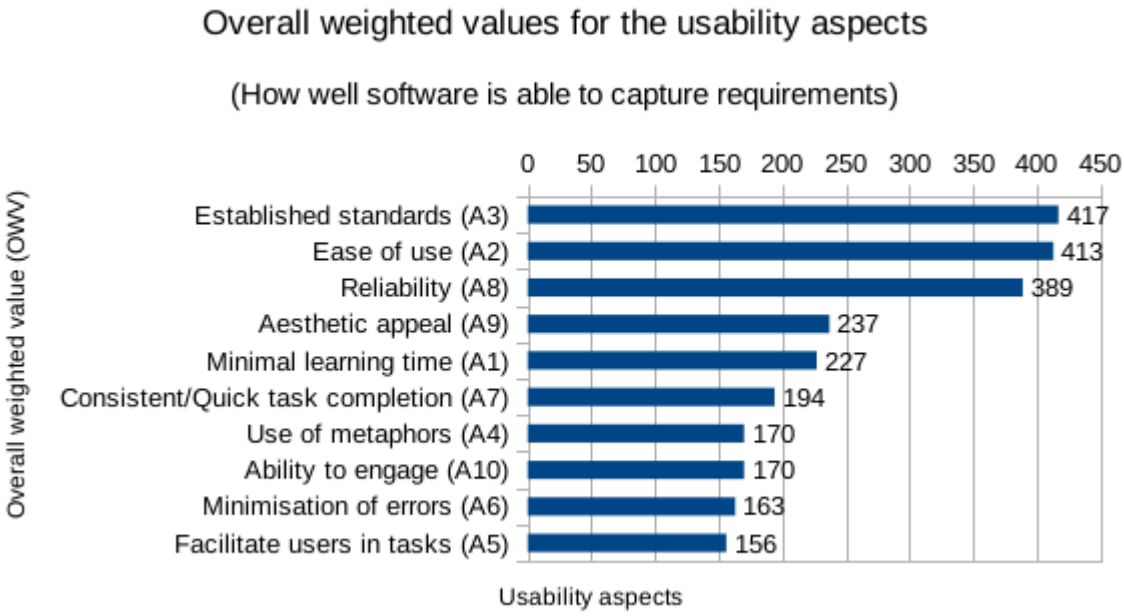


8.2.2 Ordering of the usability aspects

The final column in the above table gives the overall weighted values (OWV) for each of the ten aspects according to the weights indicated for each degree of response (see second last row). These average values (OWVs) enabled an ordering to be made of the ten aspects according to how well the requirements for them were captured overall. This is done in from most to least well captured (Figure 28). It seems that usability requirements are most well captured for the use of established standards followed by ease of use. Reliability, aesthetic appeal and minimal learning time have relative more moderate overall weighted values. The remainder aspects have lower overall weighted values with the least overall rating for A5 (facilitate users in tasks). The finding that this aspect is weakest is significant because as a usage focused methodology, UgCD is designed precisely to ensure users can accomplish all necessary tasks. The overall weighted values of the ten usability aspects are

illustrated in Figure 28, which makes it clearer that A3, A2 and A8 received the highest indications overall.

Figure 28: Overall weighted values of the ten usability aspects



The results show the perception of developers in Saudi higher education institutions and what they think about their own team's ability to capture certain usability requirements. The developers appear to be most confident that their existing software development methods follow established standards, and can ensure ease of use and reliability of the software developed. However, usability also concerns factors such as learnability and engagement, which were rated less, so some usability issues are evident.

Most notably however, the requirements for facilitating users in tasks was not thought to be well captured, and consistent/quick task completion was not rated much higher either. This is a significant finding because it appears the developers admitted a possible deficiency for which UgCD is specially designed to ensure, i.e. to facilitate users in completing tasks. If UgCD can demonstrate significant improvement in ensuring users can complete essential tasks, then these developers could be swayed into adopting UgCD, or at least some aspects of the methodology.

8.2.3 Average values and typical responses

The pattern for the ordered overall weighted values (OWVs) is also reflected in the average values calculated for each usability aspect, found by dividing each OWV by 212.

Average (mean) value = OWV / 212 (calculated to 3 decimal places)

The average values thus calculated ranged from 0.736 (for A5) to 1.967 (for A3). The data was entered into PSPP software along with the corresponding OWVs. Three aspects had average values close to the weight (2) assigned to the response 'Quite Well', namely A3, A2 and A8, i.e. these averages were within the range 1.500-2.499. All the rest had average values close to the weight (1) assigned to the response 'Reasonably well', i.e. these averages were within the range 0.500-1.499. The average values and the associated typical responses are summarised in Table 25. In addition, an overall average value was calculated as the mean value for all the aspects, which is 1.2. This mean value indicates that overall, the respondents agreed reasonably well with the question items in part A of the survey because it lies within the range for the 'reasonably well' response. The standard deviation of the set of mean values for all the items was calculated to be 0.51. The PSPP output is attached in the appendix.

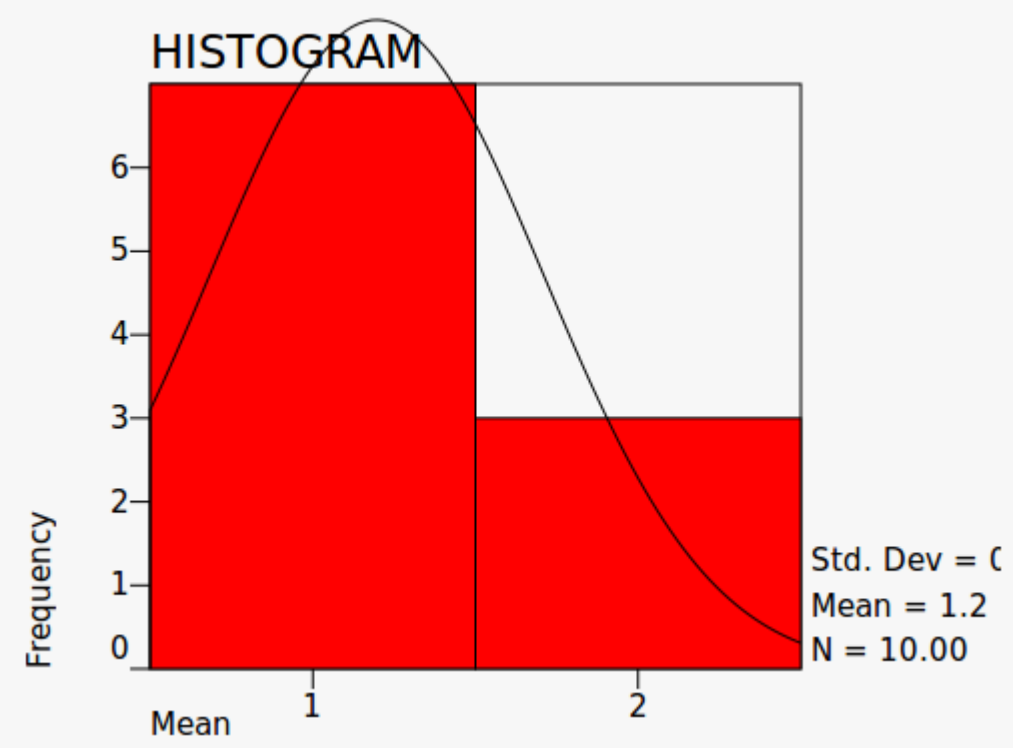
Table 25: Average values and typical responses in part A of the survey

Usability aspect		OWV	Average (3dp)	Typical response
A3	Established standards	417	1.967	Quite well (1.500-2.499)
A2	Ease of use	413	1.948	
A8	Reliability	389	1.835	
A9	Aesthetic appeal	237	1.118	Reasonably well (0.500-1.499)
A1	Minimal learning time	227	1.071	
A7	Consistent/Quick task completion	194	0.915	
A4	Use of metaphors	170	0.802	
A10	Ability to engage	170	0.802	
A6	Minimisation of errors	163	0.769	
A5	Facilitate users in tasks	156	0.736	
Range (minimum-maximum)		156-417	0.74-1.97	-
Mean		253	1.20	
Standard deviation		108.88	0.51	

8.2.4 Test for normality of data

A test for normality of data was conducted on the mean values obtained above, based on the total OWVs. The PSPP output is attached in Test for normality of the mean values (appendix). The skewness of the values was 0.84 and the kurtosis was -1.29. A histogram was superimposed on these values, as shown in Figure 29 below. The high positive value for skewness shows that most values are concentrated on the left of the overall mean with extreme values to the right, and the kurtosis value being less than 3 indicates a platykurtic distribution, i.e. a normal distribution that is flatter with a wider peak, which also makes for a lower probability of extreme values as the values are spread more widely around the overall mean (of the individual means).

Figure 29: Test for normality of the mean values



8.2.5 Exploratory factor analysis

An exploratory factor analysis was conducted in order to determine the correlation between the measures and possibly derive a factor structure, that is, reduce the number of factors that still explain the observed variance. The sample size of 212 was sufficient to conduct a

factor analysis as per Hair et al's (2007) suggestion of the size needing to be a minimum of 100 although this is considered only a fair amount according to Comrey & Lee (1992).

For factor analysis, the assumption of normality is important (Field, 2009), and this was tested and confirmed in the previous subsection 8.2.4 Test for normality of data. The skewness was found to be 0.84, which is high, but the kurtosis value of -1.29 is within the range of ±2, which was specified by Garson (2012) as necessary for the normality assumption. Notably, the skewness value is outside of this range, but the factor analysis is still justified although the solution should be treated as degraded, as advised by Tabachnick & Fidell (2012).

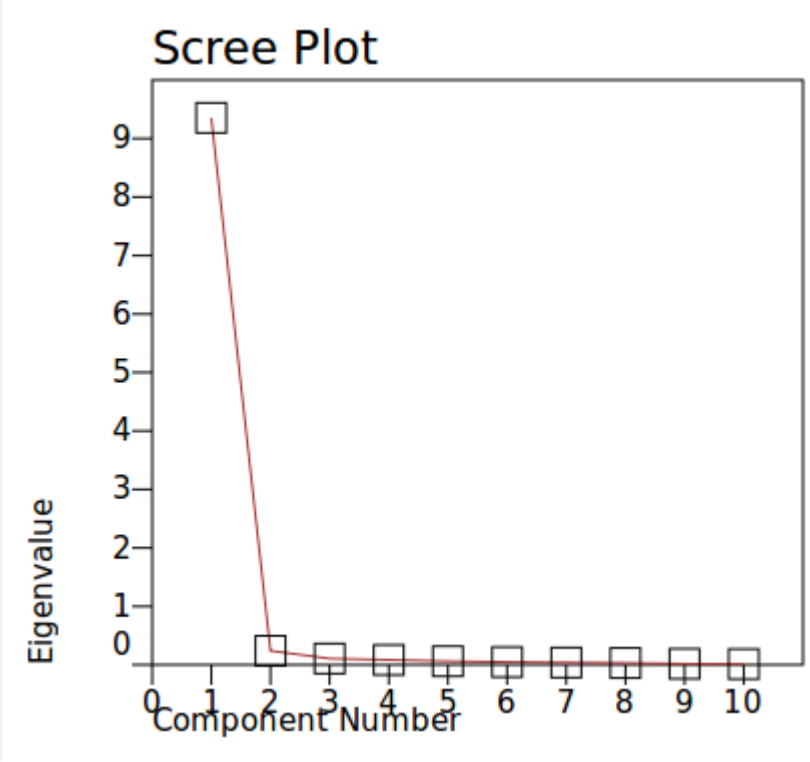
Factor analysis was first conducted for the factors identified in part A for usability aspects. The results of the factor analysis are attached in the appendix under Factor analysis and the key data explaining the total variance is summarised in Table 26. The rotation type of varimax orthogonal was chosen because it minimises the number of variables with extreme loadings on a factor. For factor extraction, Principal Component Analysis (PCA) was used as it accounts for total variance unlike Principal Axis Factoring (PAF), which only accounts for co-variation, so the original data can distributed among all extracted factors. The eigenvalue for factor retention was set to 1, as recommended by Kaiser (1960).

Table 26: Factor analysis for usability aspects (PCA applied)

Total variance explained								
Component	Initial eigenvalues			Extraction sums of squared loadings			Rotation sums of squared loadings	
	Total	% of variance	Cumulative	Total	% of variance	Cumulative	Total	% of variance
1	9.35	93.54	93.54	9.35	93.54	93.54	9.35	93.54
2	0.24	2.40	95.94					
3	0.11	1.07	97.01					
4	0.09	0.85	97.86					
5	0.06	0.62	98.48					
6	0.05	0.49	98.97					
7	0.04	0.40	99.37					
8	0.03	0.32	99.68					
9	0.02	0.18	99.87					
10	0.01	0.13	100.00					

Notably, the total of the factor loading (9.35) is greater than the value of 0.30 specified by Field (2009) for it to be significant. This is greater only for the first factor of A1 (minimal learning time). This factor alone explains 93.54% of the total variance. All the other factors (A2-A10) explain the remainder variance with A2 (ease of use) and A3 (following of established standards) accounting for a relatively greater proportion than the others. This picture is clear in the Scree plot in Figure 30 below.

Figure 30: Scree plot of factor analysis of usability aspects



8.3 Methods used to capture requirements

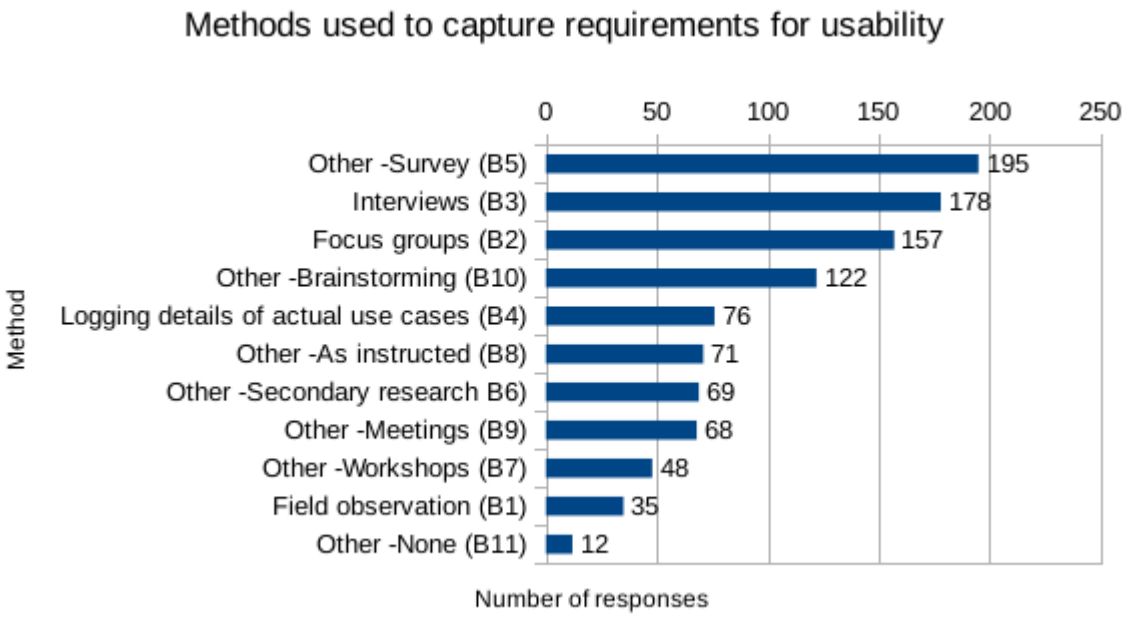
The results for the next part of the questionnaire to find out the methods used to capture requirements, are given in Table 63 (in Appendix I: Raw survey data). Only the first 4 methods were specified in the questionnaire, but the responses to the fourth open option helped to identify 6 further methods and an additional 'none' category was made making it 11 variables in total. The mean, standard deviation and t-scores for each statement pertaining to the methods used to capture requirements are given in Table 27. Notably, the mean values all range between 3 and 4 (specifically 3.20 for B11 and 3.79 for B2), which indicate the developers had a moderate perspective of recapturing requirements.

Table 27: Mean, SD and T-Scores for methods used to capture requirements

	Statement	Mean	St. Deviation	T Score
B1	Field observation	3.49	1.129	49.116
B2	Focus groups	3.79	1.130	53.346
B3	Interviews	3.58	1.240	45.989
B4	Logging details of actual use cases	3.54	1.329	42.348
B5	Other survey	3.48	1.180	46.869
B6	Other secondary research	3.26	.991	52.426
B7	Other workshops	3.31	1.050	50.105
B8	Other-as instructed	3.54	1.135	49.556
B9	Other meetings	3.43	1.134	48.079
B10	Other-brain storming	3.42	1.045	51.958
B11	Other-none	3.20	1.043	48.766

These results illustrated in Figure 31 below in an ordered way, show that the most popular method of all was the survey method followed by interviews and focus groups, and brainstorming was also popular. On the other hand, the least popular methods were workshops and field observations, and the least of all responses were indicated for 'none'. The indications for the first four listed methods (B5, B3, B2, and B10) are clearly much higher than for the other methods.

Figure 31: Methods used to capture requirements for usability



As far as UgCD is concerned, it relies heavily on frequent meetings and continuous consultations besides surveys, observations and focused enquiries or investigations. The fact that meetings (B9) are underused means that the benefits of frequent meetings may not be apparent to the developers, so there is scope for promoting UgCD among Saudi developers.

8.4 Methods used to check for usability afterwards

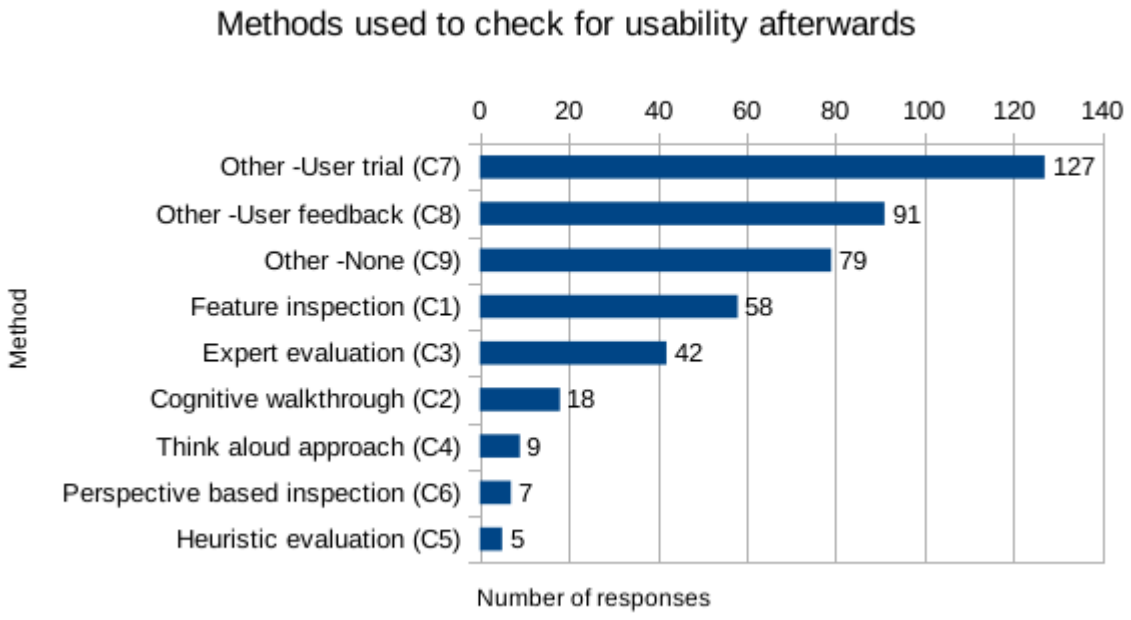
The results for the next part of the questionnaire to find out the methods used to check for usability afterwards, are given in Table 62 (in Appendix I: Raw survey data). The first 6 methods were specified in the questionnaire, but the responses to the open option helped to identify a further two methods, namely user trial and user feedback. Altogether, nine variables were defined including the option of 'none'. The mean, standard deviation and t-scores for each statement pertaining to the methods used to capture requirements are given in Table 28. The mean values range between 2.98 (for C7) and 3.66 (for C9).

Table 28: Mean, SD and T-Scores for methods used to check for usability

	Statement	Mean	St. Deviation	T Score
C1	Feature inspection	3.29	1.229	42.628
C2	Cognitive walkthrough	3.13	1.127	44.134
C3	Expert evaluation	3.23	1.146	44.856
C4	Think aloud approach	3.38	1.102	48.853
C5	Heuristic evaluation	3.10	1.305	37.811
C6	Perspective based inspection	3.53	1.022	54.886
C7	Other-User trial	2.98	1.137	41.631
C8	Other-User feedback	3.21	1.204	42.348
C9	Other-none	3.66	1.200	48.448

These results illustrated in Figure 32 below in an ordered way, show that the most popular method of all was user trials followed by feedback. On the other hand, the least popular methods were the 'Think Aloud Approach', perspective based inspection, and heuristic evaluation, and the indications for 'none' was quite high at 79.

Figure 32: Methods used to check for usability afterwards



Using the above data, an ordering can be made for the 9 variables. The indications for the

first listed method (C7) are clearly much higher than for the other methods, and three last listed methods (C4, C6, and C5) were the least used methods of all.

Usability inspection is an important component of the UgCD methodology. Some evidence of inspection is present, but there is greater reliance on trials and feedback, which are also useful for testing the software. However, for UgCD, it is important that software users are able to accomplish specific tasks whereas the trials do not seem to be focused specifically on this. Promotion of UgCD could therefore be beneficial in showing how to ensure the software that is developed is highly usable, especially in terms of users being able to carry out essential tasks.

8.5 Correlation analysis

8.5.1 Grouped correlation analysis

A correlation analysis was first attempted to see if there is any relationship between the three groups of variables:

A_m: Mean values for part A of the survey (not to be confused with item A2)

B: Values obtained for recapturing methods

C: Values obtained for usability testing methods

The analysis compared each possible pair of variables, which led to three correlation tests (A_m with B, A_m with C, and B with C).

The Likert scale used in part A provided an ordinal scale of measurement (as opposed to being a continuous variable). The same was the case with variables B and C. It was also assumed there is a monotonic relationship between each of the two sets of variables. Therefore, Spearman' rho tests were conducted, which tests for the how closely two sets of rankings agree with each other. This is in contrast with the correlation test for linear association, for which a Pearson's product moment test would have been appropriate instead.

The Spearman's rank correlation coefficient (P or r_s) was calculated for each of the three pairs of variables. The following formula was used and the calculations were done in PSPP using crosstabs:

$$r_s = 1 - \frac{6\sum d^2}{n(n^2-1)}$$

where the numerator in the fractional part includes the sum of the squared differences (d) between the pairs of ranks, and n is the total number of pairs.

At the 95% level of significance, a significant correlation is indicated by p=0.05, and the r_s values range between -1 and +1, which indicate negative to positive correlation respectively. The null hypothesis was formed that the ranks of the two variables do not covary with each other. The P values of the Spearman rank correlation were calculated for ascertaining the chance that random sampling would result in a correlation coefficient as far from zero as observed in the experiment in the case of no correlation between the two variables. Two scenarios were identified:

- Small P value: The correlation being due to random sampling can be rejected.
- Large P value: Insufficient evidence that the correlation is real and due to chance.

The PSPP output is attached in Appendix J: PSPP analysis and output. The table below summarises the main values of interest.

Table 29: Spearman's rank correlation values

Variables compared		Spearman's rho	Asymp std. error	Approx. T	Degrees of freedom	P value	No. of valid cases
A_m-B	Mean values in part A with values in part B	0	0.34	0	9	0.3	10
A_m-C	Mean values in part A with values in part C	0.58	0.16	2.86	8	0.23	10
B-C	Values in parts B and C	-0.23	0.31	-1.76	8	-0.32	11

In the case of A_m-B (0), the correlation is nil; in the case of A_m-C (0.58), there is a moderately positive correlation, and in the case of B-C (-0.23), there is a negative correlation between the two variables. Interpretation of their significances were made as follows:

A_m-B: $|rs| < 0.3$ => Acceptance of null hypothesis

A_m-C: $|rs| > 0.58$ => Rejection of null hypothesis

B-C: |rs| < 0.32 =>Acceptance of null hypothesis

Only in the case of the mean values in part A and the testing methods used in part C can it be said that there is some association between the two variables.

8.5.2 Detailed correlation analysis

In the detailed correlation analysis, each item in each of the three groups was tested for possible correlation with all other items. The three sets of correlation matrices are attached in Appendix K: Correlation matrices, which are as follows:

Set A2: Usability aspects (in Table 66, for A-J)

Set B: Usability capturing methods (in Table 67, for AA-AK)

Set C: Usability testing methods (in Table 68, for AAA-AAH)

The correlational relationships listed below are those that were found to be significant at the 95% level of confidence (for which the p-values were less than 0.05, i.e. $p < 0.05$).

8.5.2.1 Correlations for identified usability aspects

The table below lists all the pairs of variables found to be positively correlated with p-values less than 0.05 between the usability aspects.

Table 30: Correlations for identified usability aspects

Variables compared		r value	p value (<0.05)
A1 Minimal learning time	A2 Ease of use	0.456	0.010
	A3 Established standards	0.367	0.020
	A4 Use of metaphors	0.371	0.019
	A6 Minimisation of errors	0.146	0.050
	A7 Consistently quick task completion	0.261	0.045
	A8 Reliability	0.475	0.000*
	A9 Aesthetic appeal	0.549	0.000*
	A10 Ability to engage	0.474	0.000*
A2 Ease of use	A4 Use of metaphors	0.340	0.000*
	A5 Facilitating users in tasks	0.403	0.000*
	A6 Minimisation of errors	0.440	0.000*

Variables compared		r value	p value (<0.05)
	A7 Consistently quick task completion	0.303	0.030
	A9 Aesthetic appeal	0.396	0.002*
	A10 Ability to engage	0.426	0.000*
A3 Established standards	A4 Use of metaphors	0.272	0.045
	A5 Facilitating users in tasks	0.403	0.100
	A6 Minimisation of errors	0.440	0.000*
	A8 Reliability	0.232	0.044
	A9 Aesthetic appeal	0.206	0.040
	A10 Ability to engage	0.262	0.039
A4 Use of metaphors	A5 Facilitating users in tasks	0.157	0.047
	A6 Minimisation of errors	0.310	0.026
	A7 Consistently quick task completion	0.105	0.049
A5 Facilitating users in tasks	A6 Minimisation of errors	0.172	0.006
A6 Minimisation of errors	A9 Aesthetic appeal	0.136	0.030
A7 Consistently quick task completion	A8 Reliability	0.212	0.001*
A8 Reliability	A9 Aesthetic appeal	0.197	0.002

**Also significant at the 99% level of confidence*

8.5.2.2 Correlations for usability capturing methods

The table below lists all the pairs of variables found to be positively correlated with p-values less than 0.05 involving the usability capturing methods (set B).

Table 31: Correlations involving usability capturing methods

Variables compared		r value	p value (<0.05)
A1 Minimal learning time	B4 Logging details of actual use cases	0.171	0.040
	B6 Other - secondary research	0.131	0.037
	B9 Other - meetings	0.258	0.000
A2 Ease of use	B3 Interviews	0.302	0.038
	B4 Logging details of actual use cases	0.441	0.000*
	B5 Other - survey	0.159	0.011
	B9 Other - meetings	0.206	0.000

Variables compared		r value	p value (<0.05)
	B10 Other - brainstorming	0.409	0.000*
A3 Established standards	B10 Other - brainstorming	0.376	0.015
A4 Use of metaphors	B1 Field observation	0.249	0.000
	B3 Interviews	0.134	0.033
	B6 Other – secondary research	0.256	0.000*
	B7 Other - workshops	0.126	0.045
A5 Facilitating users in tasks	B5 Other - survey	0.166	0.006*
	B6 Other – secondary research	0.136	0.030
	B8 Other – as instructed	0.444	0.000*
A6 Minimisation of errors	B1 Field observation	0.146	0.020
	B4 Logging details of actual use cases	0.361	0.000*
	B6 Other – secondary research	0.150	0.017
	B8 Other – as instructed	0.178	0.005*
	B9 Other - meetings	0.251	0.000*
A7 Consistently quick task completion	B1 Field observation	0.599	0.000*
	B3 Interviews	0.225	0.000*
	B7 Other - workshops	0.225	0.000*
	B8 Other – as instructed	0.276	0.000*
A8 Reliability	B6 Other – secondary research	0.440	0.000*
A9 Aesthetic appeal	B6 Other – secondary research	0.446	0.000*
A10 Ability to engage	B6 Other – secondary research	0.159	0.110

**Also significant at the 99% level of confidence*

8.5.2.3 Correlations for usability testing methods

The table below lists all the pairs of variables found to be positively correlated with p-values less than 0.05 involving the usability testing methods (set C).

Table 32: Correlations involving usability testing methods

Variables compared		r value	p value (<0.05)
A1 Minimal learning time	C7 Other – user trial	0.129	0.041
	C8 Other – user feedback	0.231	0.000*
A2 Ease of use	C3 Expert evaluation	0.292	0.000*

Variables compared		r value	p value (<0.05)
	C9 None	0.229	0.000*
A3 Established standards	C2 Cognitive walkthrough	0.250	0.000*
	C9 None	0.326	0.000*
A5 Facilitating users in tasks	C2 Cognitive walkthrough	0.176	0.005

**Also significant at the 99% level of confidence*

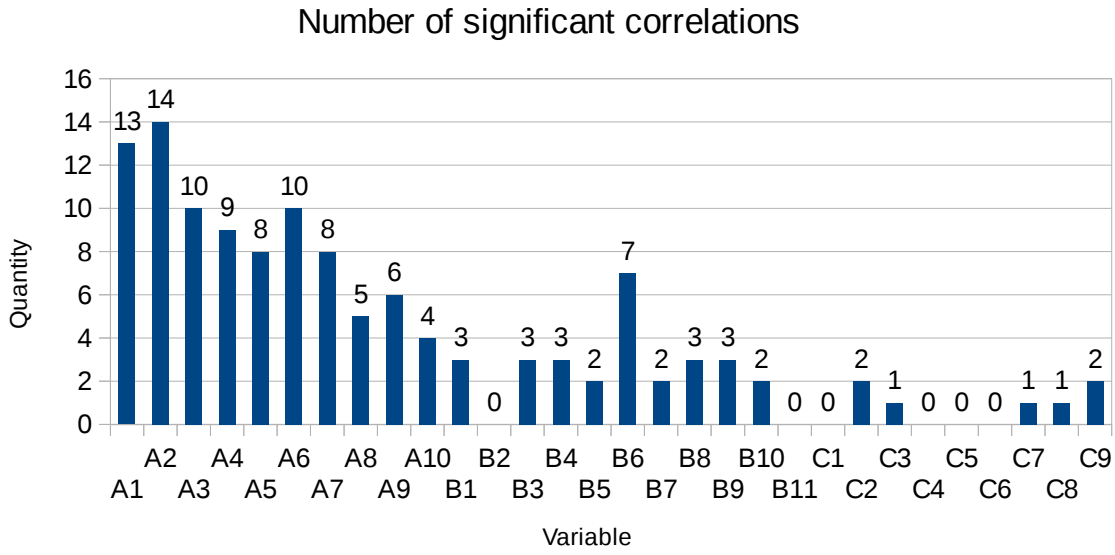
8.5.2.4 Significant correlations

The above three tables listed all the positive correlations between the 30 variables that were found to be significant at the 95% level of confidence (with p-values less than 0.05). A few pairs among these were also highly significantly correlated at the 99% level of confidence (with p-values less than 0.01), which are indicated by an asterisk (*). Table 33 below summarises the number of significant correlations for each of the 3 sets of variables, and the proportions are illustrated in Figure 33.

Table 33: Number of significant correlations for each of the 3 sets of variables

Set A(2)	Number	Set B	Number	Set C	Number
A1	13	B1	3	C1	0
A2	14	B2	0	C2	2
A3	10	B3	3	C3	1
A4	9	B4	3	C4	0
A5	8	B5	2	C5	0
A6	10	B6	7	C6	0
A7	8	B7	2	C7	1
A8	5	B8	3	C8	1
A9	6	B9	3	C9	2
A10	4	B10	2	-	-
-	-	B11	0	-	-

Figure 33: Chart of number of significant correlations



The first two variables (A1 and A2) stand out as the variables that are most correlated with other variables. They confirm the understanding that usability is fundamentally about minimising the time required to learn to use a software and its ease of use. These two are then followed in extent of importance by the next five variables (A3-A7) pertaining to usability aspects, namely following of established standards, use of metaphors, facilitating users in tasks, minimising errors, and consistently quick task completion. The remainder variables (A8-A10) were significantly correlated with at least 4 other variables.

Among the variables in Set B, variable B6 (use of secondary research) stands out the most being correlated with 7 other variables. It was significantly correlated with A1, A4, A5, A6, A8, A9, and A10, which is a wide range of usability aspects with the exception of ease of use, (A2), established standards (A3) and consistently quick task completion (A7). Notably, this item was identified by the survey respondents as a method used for gathering usability requirements, so it was categorised under 'other' as it was not pre-specified in the questionnaire. This suggests reliance on secondary research, as opposed to directly involving users for gathering requirements, is not only used widely but it is also significantly associated with many of the developer notions of usability. The third set of variables (Set C) contained the most number of variables with no correlations at all (C1, C4-6), and none of the other variables had more than or or two significant correlations.

This may suggest testing for usability is either varied or not given much prominence, or both.

8.6 Summary of survey results

A survey was conducted in which there were 212 software developer respondents from all over Saudi Arabia who responded to questions related to usability aspects, capturing methods and testing methods. A six-point Likert scale was used stretching from 'not well at all' to 'very well' for developers to indicate their responses to the first part on how well they thought their recently software developed captured usability requirements. The ten items evaluated in part A are listed in Survey questionnaire for developers, and the average values of the responses ranged from 0.736 (for A5) to 1.967 (for A3). The totals for each degree of response show that they are generally skewed towards the higher end of being well captured (Figure 26). This is especially true for 'established standards' (A3), 'ease of use' (A2), and 'reliability' (A8). On the other hand, uncertainty was greatest for 'minimisation of errors' (A6), and the negative responses were greatest for the same and most for 'facilitate users in tasks' (A5).

In short, Saudi developers are most confident that their existing software development methods follow established standards and ensure ease of use and reliability whereas other usability aspects related to learnability and engagement are rated less. Notably, requirements for facilitating users in tasks are not thought to be captured well. Also, the factor analysis revealed that 93.54% of the total variance is explained by the first factor (A1) of minimal learning time. The other factors account for the remainder variance but the most prominent amongst them are ease of use (A2) and following of established standards (A3).

As for methods used for capturing requirements, the developers appear to have a moderate perspective overall of recapturing requirements. The most popular method used is the survey followed by interviews and focus groups whereas workshops and field observations are least common. Notably, meetings (B9) are not used so much. For checking usability post-development, Saudi developers appear to conduct user trials (C7) the most followed by obtaining direct user feedback (C8). Methods such as the Think Aloud approach (C4), perspective based inspection (C6) and heuristic evaluation (C5) were least used.

The correlation analysis data for the three groups of variables mentioned above is given in

Table 29. These three groups were A_m (mean values for part A of the survey regarding usability aspects for which requirements are captured), B (methods used for capturing requirements), and C (methods used for testing for usability afterwards). This analysis showed that at the 95% level of significance, there is no correlation between A_m and B, a moderately positive correlation between A_m and C (r=0.58), and a negative correlation between B and C (r=-0.23).

The more detailed correlation analysis between each item in each of the above-mentioned three groups, which amounted to 30 variables. Numerous positive correlations were established that are listed in Table 30, Table 31, and Table 32, some of which were also significant at the 99% level of confidence. A further analysis of the quantity of significant correlations for each variable (fig29) showed that the first two variables, namely A1 (minimal learning time) and A2 (ease of use) especially, stand out as being correlated most with other variables. This confirms that these two aspects are mostly associated with the concept of usability among Saudi developers. Of the usability requirement capturing methods, variable B6 (use of secondary research) was correlated the most with others, and none of the variables relating to usability testing methods in category C were as prominent.

Chapter Nine

Interview Data and Analysis

Chapter 9: Interview data and analysis

9.1 Introduction

This chapter presents the findings made from conducting the interviews with software developers in the Saudi kingdom based on a thematic analysis of the interviewee responses. The sample characteristics and the interview dates, venues and types were described in 7.3.2 Interview sample, and the findings from the thematic analysis is presented in this chapter in sequence for each of the 11 interview questions. This includes a frequency analysis for each item to show the variations in the responses as well as identify which were common.

The responses of the interviews for each of the questions were entered into a database and a thematic analysis was conducted to help identify the main emergent themes, i.e. commonality in the different responses. The database enabled for easier comparison between the different interviewee responses for each question besides viewing all the responses of each interviewee. After visually comparing these responses, keywords and phrases with either exact or similar patterns were identified and their frequencies were noted in the analysis that follows, i.e. a list was made of the interviewees that mentioned those same or similar words and phrases. This in turn helped to identify the most and least common themes in all the responses. The most common themes were taken to be indicative of the typical responses to each of the questions.

9.2 Responses to the questions

9.2.1 Responses to question 1

Q: Please describe how you manage to capture user requirements for the software you have developed in the recent past?

Table 34 presents a summary of the key themes that emerged in the responses of the developers on the methods they use to capture requirements and the respondents who mentioned them.

Table 34: Methods used to capture requirements

Method	Interviewee	Total
Interviews	1, 9, 12	3
Focus groups	1, 3, 4, 9, 13, 18, 19	7
Surveys	1, 2, 3, 9, 12, 13, 16, 19, 20	9
Workshops	1, 5, 9	3
Observations	1	1
Prototype/Beta testing	1, 2	2
Brainstorming	5, 11	3
Meetings	5	1
Own planning/PMP	6, 7, 15, 17	4
Involves users, but method not specified	8, 10, 11, 14	4
Client instructions	18, 20	2

The most popular methods for eliciting requirements were survey and focus group, and observation and meeting were least used. These findings are in general consonance with the survey questionnaire results where the same information was asked for. However, interviews were only mentioned by three interviewees whereas the survey showed the interview method to be more popular than focus groups among the survey sample. The workshop and brainstorming methods come next. It can safely be assumed then that the main methods used to capture requirements are survey, focus group and interview, followed by workshop and brainstorming. Besides these actual methods, 4 interviewees also mentioned they have adopted the PMP method, and 4 also that they involve users but without specifying the actual method. Under PMP, control and detailed planning is paramount. Everything is scheduled, measured or evaluated. This approach to undertaking engineering projects is contrary to the principles of agile development.

The comments made in the survey analysis also apply here in that meetings are not so popular whereas they are central to the UgCD methodology. However, considering that a focus group is also essentially a meeting, this deficiency may not be that significant. Moreover, it is important that users are involved, and in this regard the findings are promising because many interviewees stressed on their involvement of users.

9.2.2 Responses to question 2

Q: What do you understand by the term usability?

Table 35 presents a summary of the key themes that emerged in the responses of the interviewees to the question of what they understand usability to mean.

Table 35: What developers understand by the term usability

Item	Component/Aspect	Interviewee	Total
1	Easy to use	1, 3, 5, 6, 9, 10, 12, 13, 16, 19, 20	11
2	Enjoyable/Likeable/Pleasurable /How it looks /Satisfactory	1, 3, 5, 9, 12, 13, 14, 19, 20	9
3	Reliability	1, 3, 9, 12, 16, 19	6
4	Able to use as one would like it /Designed as specified /Meets requirements /How well it is made	1, 3, 6, 10, 11, 14, 15, 18	8
5	How well it can be used /How well it works /Fit for use /How good it is /Adequate performance	2, 3, 4, 6, 7, 10, 11, 13, 14, 15, 17, 20	12
6	Efficient	5, 9, 11, 12	4
7	Useful	8	1
8	Easy to remember/learn	12	1
9	Compatibility	12	1

Understanding of usability mostly centres around ease-of-use (1) and reliability (3, 5) followed by enjoyability (2) and other expectations (4). By name, reliability was mentioned by only 6 interviewees, but other descriptives (in item 5), which were also concerned with reliability and performance aspects, make this the most understood overall aspect of usability along with ease-of-use. Another important aspect for UgCD however, is rememberability or learnability, which only mentioned by one interviewee although may have combined their conception of it with ease-of-use. This aspect is discussed further in the next sub-section. Generally, it is good that developers are aware of usability aspects, and specifically the need to ensure ease-of-use of reliability, but there may be a need to distinguish between general ease-of-use and learnability. If the software is well-designed and makes it easy for users to accomplish tasks, then learnability/rememberability is an important aspect of usability that awareness of UgCD can help to promote.

9.2.3 Responses to question 3

Q: How well do you think your software has been able to ensure usability in terms of how well users are able to learn to use the software?

Table 36 presents a summary of the key themes that emerged in the responses of the developers on how well they think users are able to learn to use the software they develop.

Table 36: How well developers think users are able to learn to use the software they develop

Item	Response	Interviewee	Total
1	Logical organisation	1	1
2	User trial/feedback	1, 2, 10,	3
3	Detailed planning /Use of PMP	2, 5, 6, 11, 14, 15	6
4	Familiar interface elements	3, 7, 20	3
5	Familiar procedures	3	1
6	Users are asked/consulted	4, 9, 16, 17, 18, 20	6
7	Careful design	4, 8, 9, 10, 11, 12	6
8	Continuous improvements	5, 10	2
9	Consistency	7	1
10	Efficiently made	13	1
11	Unclear	19	1

This question specifically asked about the learnability aspect of developing software. The interviewees were generally confident of ensuring learnability despite previously not identifying it by name. Again, most interviewees pointed out that this is attributed to their detailed planning, consultation with users, and also to careful design. Other main attributions mentioned were user trials or solicitation of feedback, use of familiar elements for the interface, and the rest were least mentioned. The involvement of users is a good sign, but the low attention to items 1, 4, 5, and 9 are indicative once again of a possible issue with ensuring learnability.

9.2.4 Responses to question 4

Q: How well do you think your software has been able to ensure usability in terms of how well users are able to use the software effectively?

Table 37 presents a summary of the key themes that emerged in the responses of the developers on how well they think users are able to use the software effectively that they develop.

Table 37: How well developers think users are able to use the software they develop

Response	Interviewee	Total
Involvement of users	1, 2, 3	3
Trial/Feedback	1, 2, 20	3
Detailed planning/design /Use of PMP	2, 6, 8, 9, 11, 12, 13, 14, 15, 19	10
Testing	3	1
Not specified	4, 7	2
Continuous improvement	5, 20	2
All necessary requirements ascertained /Designed as required	10, 16, 17, 18	4

For ensuring users are able to use the software effectively, most interviewees mentioned their attention to detailed planning, and four simply mentioned their requirements are ascertained or the software is designed as required. Notably, software testing was only mentioned by one interviewee, as it seems most were confident that the software they develop can be used effectively to begin with.

9.2.5 Responses to question 5

Q: How well do you think your software has been able to ensure usability in terms of how well users are able to to use the software reliably?

Table 38 presents a summary of the key themes that emerged in the responses of the developers on how well they think users are able to use they software they develop reliably.

Table 38: How well developers think users are able to use the software they develop reliably

Response	Interviewee	Total
Reliability	1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20	19
Trial/Feedback	1	1
Teamwork	1, 20	2
Planning	2, 5, 6, 7, 8, 14, 15, 18	8
Continuous improvement	5, 6, 20	3
Efficiency	9	1
Made to work as designed /Made carefully /As per requirements	12, 13, 19	3
Unspecified	17	1
Testing	20	1

With regard to reliability, all the interviewees appeared to be confident of their ability to ensure this aspect of usability. It was acknowledged as a very important aspect of usability. As interviewee 3 remarked, “Reliability is a very important consideration in software development”. And, as the first interviewee said, “We give high priority to reliability of all our software”, so reliability is also taken into consideration during the design very highly. However, for ensuring reliability, the most frequently mentioned point was that this was achieved through careful planning. As interviewee 7 remarked, “Our planning is very detailed, so the software is always going to be reliable, otherwise as I said, it would not be given to the end users.” Other interviewees, such as interviewee 10, also stressed this last point that they would not release the software until they were sure it worked reliably.

9.2.6 Responses to question 6

Q: How well do you think your software has been able to ensure usability in terms of how well users are able to to use the software in an engaging manner?

Table 39 presents a summary of the key themes that emerged in the responses of the developers on how well they think the users able to use the software that they develop in an engaging manner.

Table 39: How well developers think users are able to use the software engagingly

Response	Interviewee	Total
Reliability is more important.	1, 2, 7, 9, 10	5
Efficiency is more important.	1, 9	2
Feedback /User involvement	1, 3, 9, 10, 11	5
Effectiveness is important.	2, 5, 7	3
Usability	2	1
Engaging is important to motivate	2	1
Trial	2	1
Focus group	4	1
Engagement is important too.	1, 2, 5, 7, 10, 12	6
Learning is more important.	6	1
Unspecified	8, 13, 14	3

In contrast to the responses for reliability, engagement was not seen as important for ensuring usability on equal terms, but rather as an additional aspect. This view is reflected, for instance, in interviewee 10's comment that “Making the software engaging is a concern we *also* take into account but *not as important* I would say as reliability” (emphasis added). No interviewee denied the importance of making software engaging, although interviewee 6 came close to this position, but they gave it a low priority.

9.2.7 Responses to question 7

Q: How do you test for usability afterwards?

Table 40 presents a summary of the key themes that emerged in the responses of the developers when asked how they test for usability after the software has been developed.

Table 40: How developers test for usability afterwards

Response	Interviewee	Total
You mean after the software is finished?	1, 4, 8, 9, 10, 13	6
Testing	1, 2, 5, 6, 12, 13, 14	7
Involving users	1, 4	2
Only finish when satisfied/deemed fit /Test for usability before release	1, 3, 4, 6, 7, 9, 10, 11, 13, 14	10
Trialling	2	1
Survey	2	1
Ongoing testing/improvements	5, 9, 12	3
Detailed planning	7	1
Use special software	12 (Morae)	1
Task performance checking	12	1

Asking about testing for usability afterwards was another question that was also asked in the survey. From the interviews, it appears that this is not a common practice. In fact, many interviewees had to be prompted to clarify that the question is asking about the method that is used for testing usability after the completion and release of the software. To this, those that did test for usability, reiterated they only release software when they are confident it is completed, thereby implying no need for further testing. Nonetheless, for those that did engage in some post-development testing of usability, the interviews were not fruitful for ascertaining which methods were most popular. For this, the survey identified user trials as by far the most popular method followed by user feedback. Although these are welcome measures, the only moderate level use of methods such as feature inspection again show a potential for promoting UgCD. Notably, one specific software was mentioned, namely Morae. This is a software usability testing tool available in the market, which among other techniques uses a camera for recording non-screen interactions, and involves a survey[2].

9.2.8 Responses to question 8

Q: Are you satisfied with your existing procedures to capture software requirements for usability design?

Table 41 presents a summary of the key themes that emerged in the responses of the developers on their satisfaction with their existing procedures for capturing usability requirements.

2 See http://www.techsmith.com/morae-features.html.

Table 41: Satisfaction of the developers with their existing procedures for capturing usability requirements

Item	Response	Interviewee	Total
1	Yes, very much/definitely/certainly.	1, 2, 5, 7, 8, 11	6
2	Yes.	3, 4, 6, 10, 12, 13, 14	7
3	Spend a lot of time	1	1
4	Careful planning/gathering of requirements /Detailed procedures	1, 2, 3, 5, 7, 8, 9, 11	8
5	Involve users /users are asked directly	2, 3, 9	3
6	Software only released after satisfied	4, 7	2
7	Good team of developers	6	1
8	Adoption of quality standards	9	1
9	Involvement of management	9	1

In regard to capturing requirements, all the interviewees expressed confidence in their ability to do so, and most attributed this again to the careful planning or detailed procedures. As interviewee 1 said, “Yes, we are very satisfied because we spend a lot of time and we gather the important requirements very carefully...”. As item 1 shows, some were even very confident to a high degree. Other attributions were to involving users in the process (item 5), having a good team of developers, adopting quality standards, and involving management.

9.2.9 Responses to question 9

Q: Do you think there is scope for improving the way software requirements are captured?

Table 42 presents a summary of the key themes that emerged in the responses of the developers on whether they think there is scope for improving how software requirements are captured at present.

Table 42: Scope perceived by developers for improving how software requirements are captured

Item	Response	Interviewee	Total
1	Yes, there is always scope for improvement.	1, 4, 8, 12	4
2	Depends on available time.	1, 4	2
3	Maybe/Possibly.	2, 10, 11	3
4	Spend a lot of time already.	2, 8, 9	3
5	Yes /Receptive	3	1
6	Depends on investment.	4	1
7	Careful planning/requirements gathering	5, 6, 7, 12, 13	5
8	Continuous improvements	6	1
9	Satisfied as it is	7, 9, 10, 11, 14	5

When asked about the scope for improving the way the requirements are captured, most interviewees either stated they were satisfied with the way things are, or reiterated the point about doing careful planning already. On the other hand, as item 1 shows, a few admitted to there always being scope for improvement and were therefore receptive to suggestions.

9.2.10 Responses to question 10

Q: Would you be willing to improve this requirements capturing process if it is possible to do so?

Table 43 presents a summary of the key themes that emerged in the responses of the developers on whether they would be willing to improve their requirements capturing process.

Table 43: Whether developers would be willing to improve their requirements capturing process

Item	Response	Interviewee	Total
1	Yes, willing /Certainly /Why not?	1, 3, 4	3
2	We should always strive to be more efficient.	1	1
3	Management dictates what we do /We would have to ask the management.	2, 7, 8, 11	4
4	Team approval would be needed.	3	1
5	Already experienced /We continuously improve our software.	5	1
6	Already satisfied the way things are /No need to add more elements. /Management unlikely to approve	6, 9, 10, 13, 14	5
7	Depends on project at hand	12	1

When asked about their willingness to improve the requirements capturing process, most interviewees (item 6) again mentioned they were satisfied with the way things are, and only a few (item 1) showed more willingness to try something else. In those cases in which there was no receptiveness, a common reason pointed out was the restrictions imposed by the management (item 3) as they determine what procedures are carried out and any changes would have to be approved by them. As interviewee 7 made it clear, “Our management has defined the process very well for us. Personally, I am open to ways for improvement, but for in-house development, any changes to the existing procedures would have to be approved.”.

9.2.11 Responses to question 11

Q: Are you aware of Usage-Centred Design and its possibilities for improving software usability?

Table 44 presents a summary of the key themes that emerged in the responses of the developers on whether they were aware of UgCD and its possibilities for improving software usability.

Table 44: Awareness of UgCD among the developers

Item	Response	Interviewee	Total
1	No.	1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14	12
2	No, don't think so	3, 10	2
3	No, but I know there are many methods.	1, 2, 3, 6, 9	5
4	We use PMP.	1, 5, 7	3
5	We have to follow management's instructions.	1, 14	2
6	Satisfied the way things are	2, 6	2
7	Heard of User-Centred Design (but not UgCD)	3, 4, 12	3
8	(Was interested to be briefed about UgCD)	3, 9, 12	3

The final question sought to find out if any developer was even aware of UgCD referred to them as Usage-Centred Design. Importantly, NO developer had heard of UgCD before. However, some acknowledged there are many methods (item 3), and some reiterated their satisfaction with the way things are (item 6). Three interviewees pointed out that they had heard of User-Centred Design (UrCD), but not UgCD (item7), and only three (item 8) were interested to learn about UgCD and were therefore briefed about it.

9.3 Summary of interview findings

Interviews were conducted with 20 software developers working in higher education institutions in Jeddah, Riyadh and Taif. A thematic analysis with the aid of a database helped identify the most and least common themes present in the interviewee responses, and the interviews as a whole helped to gain insight into the thoughts of the software developers to shed light on their developmental practices. Notable findings were that:

- Survey and focus groups are the most popular methods used for eliciting requirements whereas observation and meeting were least used (Q1).
- Usability is mostly understood as relating to ease-of-use and reliability (Q2).
- Learnability is not considered so much (Q2), although it was affirmed as being considered by relying on detailed planning, through careful design and by consulting users (Q3).
- The reliance on detailed and careful planning was reiterated prominently or otherwise made apparent especially for Q4, Q5, Q8

- Usability is rarely tested post-development (Q7).
- Saudi developers are generally content with the current procedures for ensuring software usability (Q9, 10) and not so willing to try another method for improving the way requirements are captured (Q10) due to managerial constraints with only a few exceptions.
- Usage-Centred Design was totally unheard of (Q11) although some acknowledge there are many other methods, and only three (15%) expressed an interest in learning about it.

Chapter Ten

The UgCD Trial

Chapter 10: The UgCD trial

10.1 Introduction

This chapter details the outcome of the UgCD trial that was held as an opportunity to demonstrate UgCD during the phase of re-capturing requirements for an existing software. It begins with an outline of the background and context within which the trial took place. The outcome of the meeting is then described in detail in terms of the users, roles and tasks that were identified, the focus of the meeting, and the requirements that were captured and the aspect to which they pertained through comparing the previous and present situation. This is followed by analysing the measures of completeness and preciseness of the two sets of requirements, and showing the software changes that were implemented as an outcome of the trial.

10.2 Background and context

10.2.1 Selection of the institution for the trial

As the interview findings showed, particularly for question 10, most interviewees were reluctant to try another method to their existing practices. Most in fact were restricted by the control of their management. From question 10, in the previous chapter, it is evident that developers had limited control over the implementation of new usability testing as management and team approval were needed. Another key factor highlighted by the developers was that they were satisfied with the way in which their current system operates. Therefore developers 1, 3, 4 and 12 were the key targets of the implementation trial. The lack of team approval in the case of number3, and the lack of time in the case of numbers 1 and 4, resulted in the testing of UgCD as part of a project currently being worked on by Developer number 12. Interviewee 12 agreed to discuss the request with his team. However, certain conditions were made that restricted the demonstration of UgCD as originally intended; in its entirety. Details of these limitations are given further below in 10.2.3 Limitations of the trial.

10.2.2 Background and comparisons

As it happened, the developers were working on improving the interface of an online exam system at the time, so UgCD was applied for this purpose. The system chosen for the

implementation was therefore an online examination system. This system was developed in recent years to provide a platform for the teachers to conduct tests and exams by having students undertake them using an online system.

Prior to the UgCD implementation, the developers had already attempted to make some improvements, i.e. without UgCD. For this purpose, they relied on the System Usability Scale (SUS)[3] and informally consulted some teachers and students, but no meeting was held. The developers informed the researcher that for the past 3 years, and especially during the previous 14 months, they had been relying on questionnaires to bring about improvements. SUS is claimed to provide a "quick and dirty" measure of usability, and the tool consists of 10 questions, which are answered using a 5-point Likert scale. A copy of this scale is attached in Appendix M: SUS questionnaire. Based on the feedback gained, some improvements were therefore already being made by the developers prior to applying the UgCD methodology using SUS as well as obtaining feedback from teachers and students in an informal setting.

However, the use of the UgCD helped to assess, for instance, (a) possible further improvements that could be made to the system (b) efficiency of improvement when compared to the SUS and the informal feedback. The idea of UgCD was therefore welcomed as an opportunity to try a different method, and to also consider software usability whereas previously the concern was mainly with providing functionality. The developers also pointed out that the improvement process had been slow using questionnaires, that the development mainly involved adding, remove or returning features, and that the software had become very large and complex over time.

In the analysis that follows, two comparisons were made for UgCD. The list of requirements captured through UgCD was first compared with the occasion when the requirements were originally captured at the time of construction. It was then also compared with the above-mentioned occasion when an attempt was made to improve the interface without using UgCD. These three occasions are referred hereinafter as T1, T2, and T3, as follows:

- T1 – Capture of software requirements at the time of construction of the original system

3 Available at http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

- T2 – Attempt to capture software requirements to improve the interface prior to the UgCD implementation (using SUS)
- T3 – Capture of software requirements to improve the interface during a meeting under a UgCD implementation guided by the researcher

10.2.3 Limitations of the trial

The UgCD methodology was trialled for recapturing requirements in a restrictive and controlled way than originally hoped for, at a higher education institution in Saudi Arabia that prefers to remain anonymous. The implementation of the trial was limited in the following main ways relating to the researcher and software users:

- Researcher: The researcher could not be present in person due to cultural restrictions that demand strict gender segregation, so instructions were given to and carried out by the team of developers who are all males.
- Software users: A few teachers were present in the meeting, but not students (who were consulted through questionnaires instead).
- Time constraints: The implementation was restricted to a single meeting, so there was only one iteration without any further refinement of the requirements.

The fact that students were not involved in the process was a major deficiency. It was not only that it was deemed to be inconvenient to arrange to include students in the meeting, but some disagreement was also expressed by some developers who believed it to be unnecessary to involve students in a meeting. They were reluctant to do so, so permission to involve students was not therefore granted. This is despite pointing out the importance of involving students as they are likely to be the main and arguably the most important users of the software, and that this would be in line with the approach of student-centred learning. Previously, the development team was accustomed to only involving students in so far as gathering their feedback after completion of development for possible future improvements. Although it was not permitted to involve any students during the meeting, the participation of teachers was permitted.

Also, the cultural restriction of strict gender segregation limited the way in which the trial was actually carried out. The researcher was not permitted to be present openly during the trial because all of the developers in the team were males. Instructions had to be given

instead on what to do, and how to carry out the requirements gathering, which was done by the regular development team without active involvement of the researcher. Extracts from the relevant portions of this thesis were given for the purpose. The researcher did also keep in touch during the implementation, and provided inputs on the process as required and in order to clarify the process.

Due to the fact that the software had already been developed, the requirements were recaptured rather than captured anew, i.e. for an existing rather than a new software for the purpose of improving it. And, the time constraints meant that only a single iteration for recapturing requirements was made during the single meeting that was held, so there were no further refinement of the requirements as required under UgCD. The lack of time was also associated with the need to launch the improved interface in the current academic year ready for students to use. There was also a related budget constraint, which contributed to the face that only a single meeting could be held, and a single iteration made. The developers also pointed out they had limited time to update the user guides for the new interface.

10.2.4 Context of the trial

Although the researcher advised on how to implement UgCD, the process itself was carried out by the regular development team with some modifications, which were outlined above. The way each main instruction given was actually implemented is described in Table 45. The development team comprises of 3 programmers, a designer and graphics artist, and an additional technical assistant. All five of them were present during the implementation.

Table 45: Differences between the instructions and the implementation

Instruction (as defined by UgCD)	How UgCD was actually implemented
Identify users, roles and tasks.	The participants made a detailed assessment of the users, their roles and tasks.
Modelling of task cases.	Task modelling was undertaken by a single developer.
Agenda prepared jointly in advance.	No agenda was prepared in advance.
To share the model with users for feedback.	A formal meeting was held between developers and teachers (not students) who were asked for their feedback.

Focus on usability characteristics throughout.	At every stage of the implementation, the usability characteristics were kept in focus.
To arrange an initial face-to-face meeting	An initial face-to-face meeting was arranged and held between the developers and the (teacher) users.
To hold frequent meetings.	Only one meeting session was held.
To hold frequent consultations	No further consultations were reported.

The fact that the trial consisted of only a single session, and there were no further consultations, meant that none of the further techniques were applied such as making sharply focused enquiries, follow-on investigations, observations or surveys. Also, no agenda was prepared in advance. A pre-prepared agenda is important because it would have provided the context necessary to facilitate the collaboration. The extent of collaboration was therefore very limited, and the process was more formal and shorter than advised. During the meeting, there was evidence however, of some brainstorming and open interaction and discussion between the developers and teachers.

With reference to the stages identified of the requirements gathering and analysis process outlined in Table 19, only the first stages were carried out, i.e.: (1) role/task identification, (2) task modelling, and (3) requirements capturing. The remaining two stages, namely (4) requirements testing, and (5) requirements refinement were unable to be carried out. Requirements testing (stage 4) would have required more iterations and therefore more time whereas only one iteration of the requirements recapturing process could be carried out. As for the final stage (5), this would have required further consultations and meetings, which again was not possible due to time constraints. The implementation therefore had to be restricted to a single meeting, and there were no further opportunities provided to users for mutual exploration and negotiation, as is required under UgCD beyond the initial meeting.

In UgCD, the purpose of the face-to-face meetings, ongoing consultations, and discussions is to gather information, build the model of task cases, explore and negotiate, and to then approve and validate the requirements gathered. If this has been an ideal implementation of UgCD for capturing the requirements, then all of the aforementioned procedures would have been carried out, as also indicated in the left column in Table 45. Nonetheless, the outcome of the meeting, despite the constraints, is described in the subsequent subsection.

10.3 Meeting for recapturing requirements

10.3.1 Identification of users, roles and tasks

The meeting was focused on finding ways to improve the interface of the online exam system at a higher education institution used by its students, which was the current project being worked on. As mentioned previously, the attendees of the meeting comprised only of developers and some teachers although the interface being redesigned was mainly used by students. The initial phase of UgCD implementation helped to identify the users of the exam system as developers, administrators, teachers and students whose roles were identified as follows:

- Developers – Their roles are to devise the system and to provide guides for administrators and other users for using it.
- Administrators – Their roles are to monitor the system and support other users (teachers and students) in case of any problems.
- Teachers – Their roles are to create or prepare exams for the students to take, and to then mark the exams once they are taken by the students.
- Students – Their role is to take the online exam.

Although there are also other stakeholders, such as the university management, software testers and exam controllers, the study was limited to the main end-users, namely teachers and students, who can be seen as the main stakeholders for whom the software has been designed and who are in a position to derive the most benefit from it. Under each role, which are two for each user (except students who have one role), there are between 2 and 5 categories of tasks, which are listed in Table 46 below. The UgCD notation used for modelling the activities is attached in Appendix P: UgCD artefacts used during the trial.

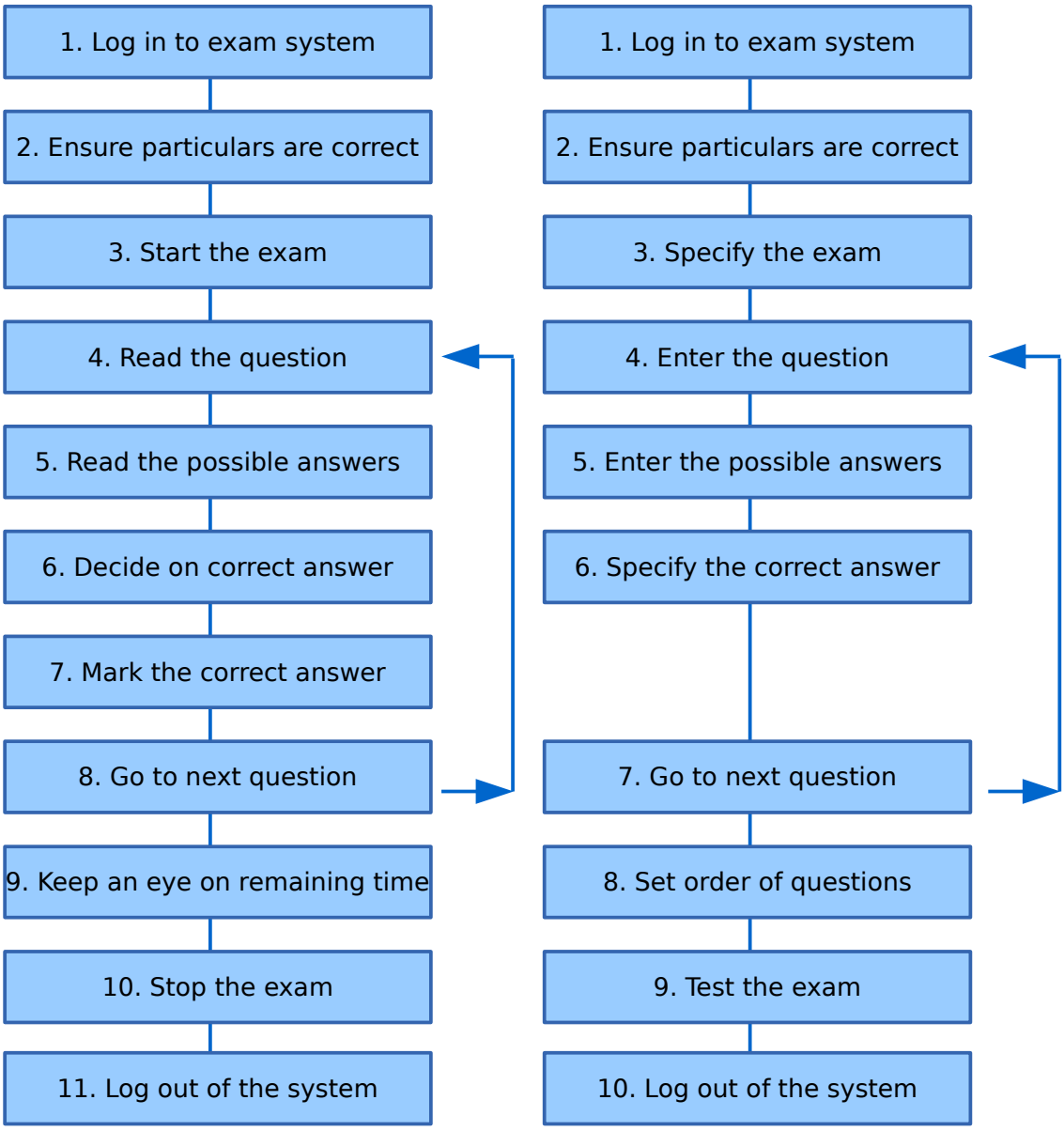
Table 46: Identified uses, roles and tasks

User	Role	Task
Developer	Devise exam system	Ascertain user needs/requirements
		Design and develop system
		Test for user acceptance of the system
		Monitor and improve system based on user feedback
	Guide administrators	Maintain updated user guides to provide information on system, its use and steps for troubleshooting
		Demonstrate use of the system
		Respond to enquiries from teachers/students regarding use of the system
		Coordinate with developers with respect to making changes based on usability requirements
Administrator	Monitor exam system	Check for uptime/reliability
		Check for cheating
		Check for use
	Support users	Provide technical support to teachers
		Provide technical support to students
		Provide user feedback to development team
Teacher	Devise/Check exam	Log in/out
		Input questions
		Set order of questions
		Test exam
	Mark exam	Check answers/responses
		Work out score
		Conduct analysis
		Share student scores with administrators to compute overall results and monitor standards
		Prepare report
Student	Take exam	Log in/out
		Read/Answer question
		Watch remaining time
		Check own results online

The key end-users of interest here are teachers and especially students. In the case of students, the flow of tasks in this system comprises of 11 sub-tasks, and as there are

similarities with the role of devising exams for teachers (comprising of 10 sub-tasks), both are illustrated side by side in Figure 34.

Figure 34: Flow of tasks in the online EMES exam system at KAU for teachers (right) and students (left)



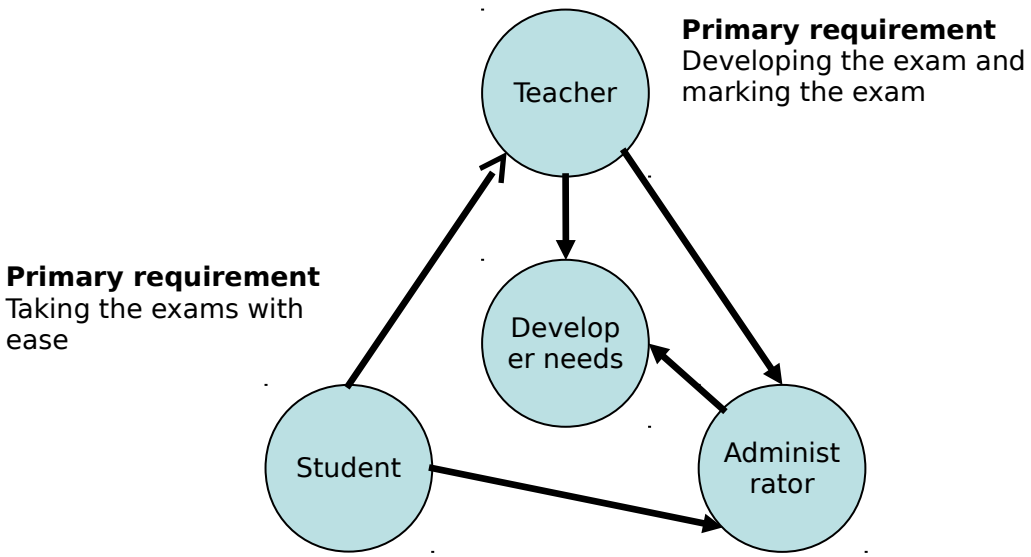
As already noted, students themselves were not present at the meeting so their role and tasks were identified by the developers and teachers who were present. In the case of teachers as end-users, they have two distinct set of tasks, which are to prepare the exams in the first place, and then after students have taken those exams, to mark their exams. The actual scoring is of course done automatically by the exam system, but the role of the

teachers in marking is to check the scores, interpret the analysis, and to generate the reports for each of their students through using the interface.

Previously, such an extensive list of users, roles and tasks had not been made, so UgCD, in which this process is an integral part of the overall UgCD methodology (Figure 18), proved to be very helpful in being clear about these. In particular, it also helped in later gathering the requirements because in so doing, the meeting participants were able to focus on requirements pertaining to each user, role and task. Moreover, by involving users in the development process, it became possible for the developers to better understand their requirements, and to prioritise their development work based on those identified user requirements.

Figure 35 below illustrates the interaction that took place between the important stakeholders, their prioritised needs, and their ability to convey those needs to the developers of the system. Notably however, although the administrators and teachers have a direct channel to convey their requirements to the developers, there is no such direct channel available at present for students other than the indirect feedback given through the SUS questionnaire.

Figure 35: Links between user requirements and communication channels to developers



10.3.2 Focus of meeting

The meeting was carried out among representatives of administrators and teachers as the users and stakeholders besides the team of developers. The administrators were senior teachers, and there were 11 of them altogether. The development team had five members, so there were 16 meeting participants in total. Throughout the meeting, a focus was maintained on usability by firstly briefing the meeting participants about UgCD and the aim of the meeting, which was to improve the usability of the interface, and being concerned through with the specific chosen aspects of usability.

It was pointed out that there were no procedural difficulties with the existing system as the students were already able to successfully carry out their tasks. In recent months, feedback from students showed that students had no negative views of the system based on the SUS tool and procedural parameters. However, they did indicate a need for more user friendly features in the interface, and for improvements to be made to the efficiency of the system to make it be more responsive and quicker to operate. The developers also pointed out that students recommended a better help system for dealing with potential problems. These points were reflected upon during the meeting.

The attention during the meeting was therefore on finding ways to especially improve the system in three main ways: (1) to improve the efficiency of certain operations, (2) to enhance the look of the student interface to make it easier to use for students, and (3) to ensure the use of the system can be learned quickly by its users. In terms of usability therefore, the focus was especially on the aspects of learnability, efficiency and user satisfaction in terms of aesthetics of the design. Learnability and efficiency in this case were important because they concerned how well the students are able to identify the key elements of the interface and learn to use it quickly, which are especially important issues for students using the system for the first time.

This contrast in the previous and new areas of focus (and neglect or less attention) is highlighted in the table below. It presents a comparison of the two areas that were being focused on or neglected during the pre-UgCD phase and during the UgCD implementation. It is evident that the initial focus was on providing the necessary functionality and reliability to obtain a working piece of software without resulting in any errors or delays. However, little or no attention was given and no substantial efforts were made to work on aspects related to aesthetics of the design, on enabling students to easily understand and

learn to use the system, or to user satisfaction in general. In contrast, the new set of usability requirements did not delve too much on reliability, as it was generally satisfactory although some efficiency related recommendations were made. Moreover, the focus was on improving learnability and attractiveness of the interface.

Table 47: Main areas of focus and neglect in the previous and new set of usability requirements

	Previous usability requirements	New usability requirements
Main areas of focus	-Providing the necessary functionality and reliability	-Improving learnability -Improving visual clarity -Improving attractiveness
Main areas of neglect (or less attention)	-Attractiveness or user satisfaction	-Efficiency and reliability (as these were considered adequate already)

10.3.3 Identification of aspects and requirements

Problem areas, aspects and elements

The assessment of the needs of students was largely gathered from indirect feedback rather than through face-to-face consultation. The requirements for students to carry out the aforementioned tasks could not therefore be identified strictly as per the guidelines issued by the researcher, so it was not possible to test for completeness and preciseness of the usability requirements as originally intended. Instead, the brief meeting enabled the teachers to point out what they perceived to be possible areas that could be worked on to improve the interface. The discussion between the attendees resulted in identifying the following problems with the old interface:

1. There were no icons to help identify important elements quickly. Instead, the teachers and students had to rely on hyperlinks to identify the necessary elements and click on the same.

2. The font used was a bit too small to read easily. Teachers felt that setting the exam questions and formatting the same was difficult.

3. There was little use of colour which made the old interface look unattractive. No efforts were made to differentiate between questions, page elements, and queries, or troubleshooting tools.

4. The interface was used for a timed test, but the remaining time indication was not prominent. This was a key complaint made by students, and administrators were aware that some students were unable to keep track of time whilst taking the exam under time pressure.

The following are the specific usability requirements that were identified by the attendees of the meeting:

- Learnability aspects: Improve learnability by highlighting important elements through the use of colour, making the text bold, and by using icons so that the elements can be identified quickly, especially the remaining time indication.
- Readability (visual clarity) aspects: Make the important text more readable at a comfortable distance by increasing its size and using a clear font.
- Attractiveness (user satisfaction) aspects: Make the whole interface look more attractive by using different colours as the old interface looked too bland and dull.

The developers did not have a copy of the old set of requirements, but they recalled that they mostly related to ensuring basic functionality and reliability of the software. As pointed out, the main aim was to simply and quickly develop a working piece of software that could be used to administer their exam system online without any 'bells and whistles'. None of the above usability requirements, which deal with making usability improvements mostly related to aesthetics, were mentioned in the old list, so in terms of completeness with regard to usability requirements, the new list is more comprehensive. The old set of requirements included the following important elements:

- Information/Display:
    - The type of test and student's particulars must be clearly displayed so that the user knows it is the right test.
    - Display: Boxes to display the question and the answer options clearly.
    - Number of questions and amount of time remaining.
- Navigation: User must be able to move easily and quickly to the next question.

List of captured requirements

In Table 48, all the known previously and newly identified requirements are listed under each selected usability aspect and in a way that shows a contrast between the two. In addition, this includes functionality as another set of essential requirements given that previously the focus was mainly on this type of requirement rather than on usability. The usability requirements are grouped under the five selected types of usability requirements, namely learnability, rememberability, efficiency, reliability and user satisfaction (as specified in 1.5 Research questions and hypotheses). In the case of functional requirements, they are further subdivided according to whether they pertain to teachers or students, and for brevity only those requirements of relevance to these two categories of users are listed.

As mentioned earlier, the comparison shows that the focus during the recent phase of modifications (during which UgCD was applied) was on improving usability in terms of learnability, rememberability and user satisfaction. It seems the developers and teachers are satisfied with the reliability and efficiency of the system as it is and did not feel the need to make any further improvements in this area, at least for the time being. Notably, there are no overlapping requirements, i.e. no requirements that were specified both previously and repeated again. This may be because of the attention to the other requirements or a genuine satisfaction with the way those requirements were implemented when they were first specified. In short, whereas previously the requirements dealt with ensuring basic functionality, reliability, and efficiency, UgCD helped to bring about improvements in the more cognitive and aesthetic aspects of software design.

Table 48: List of requirements pre- and post-UgCD

Aspect	**Specific requirement**	**T1 (Old)** (Original)	**T2** (Without UgCD)	**T3 (New)** (Under UgCD)
Functional Requirements				
Functionality for teachers	Display type of exam.	✓		
	Display teacher's particulars.	✓		
	Display area for entering question.	✓		
	Display area for entering answer options.	✓		

Aspect	Specific requirement	T1 (Old) (Original)	T2 (Without UgCD)	T3 (New) (Under UgCD)
	Display number of question.	✓		
	Allow to copy-paste question from another document instead of having to type it.		✓	✓
	Allow to print out the questions entered.		✓	✓
	Allow a 'print screen' function.		✓	✓
	Allow teachers to conduct a trial of the exam made for testing.		✓	✓
	Allow for questions to be categorised by different types		✓	✓
	The feedback report should contain more detailed analysis to identify students' weaknesses.			✓
	They system should be capable of aggregating responses to describe patterns.			✓
	Allow to specify the exam date and not let students change it.			✓
Functionality for students	Display type of exam.	✓		
	Display student's particulars.	✓		
	Display question.	✓		
	Display answer options.	✓		
	Display number of question.	✓		
	Display number of questions remaining.	✓		
	Display time remaining.	✓		
	Password login for students needing to change to another computer desk			✓
	Include essay type questions			✓
Usability Requirements				
Reliability	Students should be able to undertake the exam without any problems.	✓	✓	
	Perform sound check to make sure sound is working prior to starting exam.			✓
Learnability	Highlight important elements using colour.			✓
	Highlight important elements using bold text.		✓	✓
	Increase font size of main heading.		✓	✓
	Use another clearer and larger font.			✓
	The help system (for teachers and also students) should be made context-aware instead of presenting the list of contents.			✓
	Develop a simple user documentation for students so they can be prepared before the exam.		✓	✓

Aspect	**Specific requirement**	**T1 (Old)** (Original)	**T2** (Without UgCD)	**T3 (New)** (Under UgCD)
	Make log in and log out screens in both English and Arabic			✓
Remember ability	Use icons to represent elements for ease of identification.			✓
	Represent the time remaining information using a timer.			✓
	Display photo of student		✓	✓
Efficiency in Use	Provide navigation system for moving to the next question.	✓		
	Allow students to quickly jump to another question without having to go through the back/next buttons.		✓	✓
	Allow the keyboard navigation and tab keys to be used by students to quickly navigate the interface without having to rely on the mouse.			✓
	Reduce the number of clicks required in the teacher's interface (currently 5) to get to a particular student's previous exam record.			✓
User Satisfaction	Make interface more colourful using different colours.			✓

Suggestions for improved functionality

As regards functionality, the inclusion of functional requirements was not originally intended to be written, as the focus of the study is on usability requirements. But the requirements that were functional make up a large proportion of the total, and several new functional requirements were captured, so they are also listed in the above table separately from the usability requirements. There is a notable difference in the way functional requirements were captured previously and presently. Previously, they were captured for both teachers and students, but this time new functional requirements were only captured for teachers and no more were captured for students. This suggests there is a general satisfaction with the functionality of the student side of the system.

A few new functional requirements were specified for the teachers side in order to make further improvements for and by the teachers themselves. For instance, the teachers requested to be able to copy-paste questions from another document into the exam system instead of having to type it, to all questions and the whole screen to be printed, to be able to categorise the questions, for the system to provide automatic greater feedback, such as to identify potential academic weaknesses in students, and by aggregating responses to give

an overall picture, and to all teachers to specify the exam date and not allow students to change it. Another key functionality that was suggested was the ability to share the reports generated with the administrative office without having to use any additional software. Similarly, it was felt that being able to share the overall aggregated exam scores within the department would help to monitor student performance as a whole and target areas of learning that were indicated as common weaknesses.

One important new functionality requested for students was to expand the capability of the existing password login feature so as to allow students to change to another computer desk in case they needed to do so. It may be true that if students were present at the meeting then further new functionality requirements would have been specified by them and for themselves. Previously, the students requirements were specified by the developers.

10.4 Completeness and preciseness of the captured requirements

10.4.1 Number of requirements

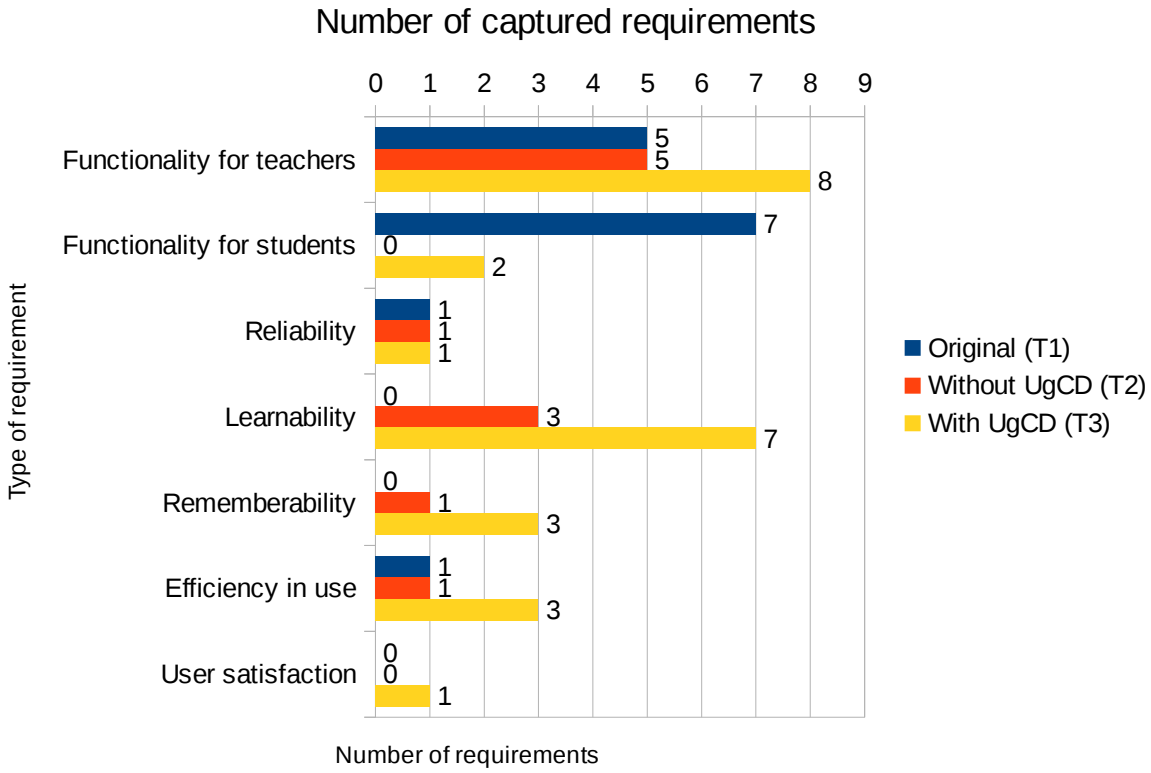
Table 48 above gave a complete list of the captured requirements, and it is apparent that the range of functionality and usability requirements are mutually exclusive between the two (previous and new) phases. The total number of requirements in this table are summarised with a breakdown by type/aspect in Table 49, and illustrated in the bar chart in Figure 36 below. Altogether, 25 requirements were captured at the UgCD meeting, which compares with 14 previously when the interface was constructed, and 11 when the improvement attempt was made without UgCD. This gives a total of 39 requirements although the previous (T1) list as given may not be exhaustive, as the requirements pertained to all the essential requirements identified originally to construct the interface whereas the new lists (T2 and T3) pertain to improvements.

Table 49: Breakdown of captured requirements

Type of requirement	Aspect of functionality /usability	Number of requirements (T1 at construction, T2-3 at improvement)			Total distinct requirements
		T1	T2	T3	
Functionality	Functionality for teachers	5	5	8	13
	Functionality for students	7	0	2	9
Usability	Reliability	1	1	1	2
	Learnability	0	3	7	7

Type of requirement	Aspect of functionality /usability	Number of requirements (T1 at construction, T2-3 at improvement)			Total distinct requirements
		T1	T2	T3	
	Rememberability	0	1	3	3
	Efficiency	1	1	3	4
	User satisfaction	0	0	1	1
Total number of requirements		14	11	25	39
Number of usability requirements		2	6	15	17

Figure 36: Number of captured requirements



10.4.2 Measures of R1, R2 and R3

With reference to the categories R1, R2 and R3 specified in 6.7.2 Adopted measures, the number of usability requirements under each of these categories are as follows:

- R1 (Usability requirements captured under UgCD but not captured already): all requirements related to learnability, rememberability and user satisfaction (15 specified for T1-T3, and 10 for T2-T3)).

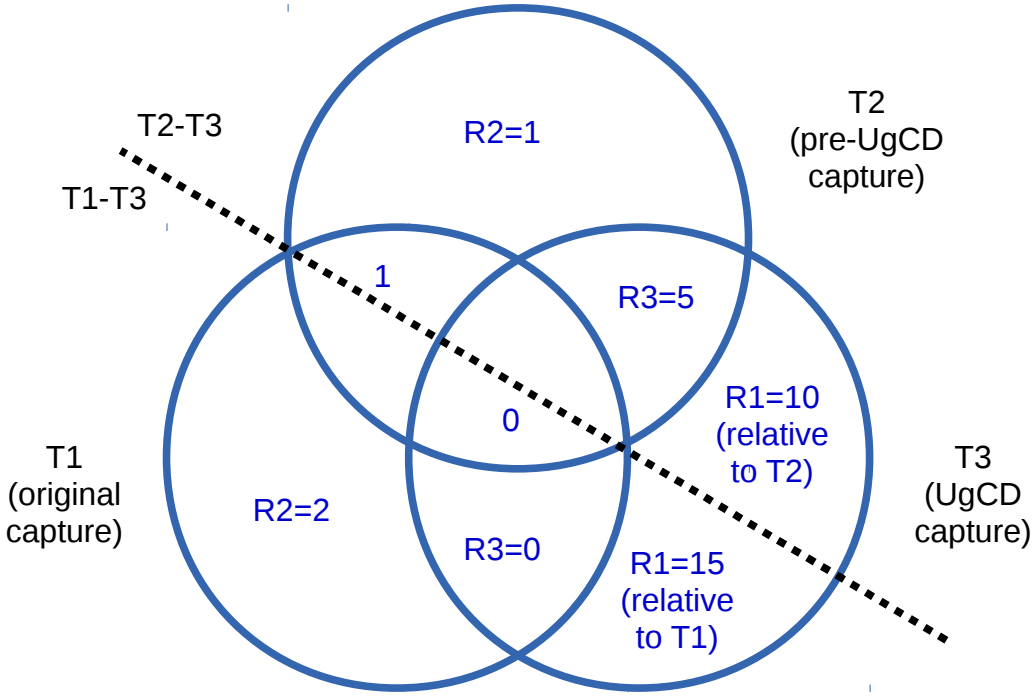
- R2 (Usability requirements captured before but not under UgCD): all requirements related to functionality, reliability and efficiency in use (2 specified for T1-T3, and 1 for T2-T3).
- R3 (Usability requirements captured by both methods): none (empty set) for T1-T3, and 5 for T2-T3.

Table 50: Summary of R values for comparisons of T1-T3 and T2-T3

R	**Description** (for usability requirements)	**R value**	
		T1 vs T3	**T2 vs T3**
R1	Captured under UgCD, but not originally or without UgCD	15	10
R2	Captured originally or without UgCD, but not under UgCD	2	1
R3	Captured on both occasions	0	5

Besides the above defined values for R1, R2, and R3, it may also be noted that one requirement was capturing during T1 and T2, but not T3, and that no requirements were captured on all three occasions (at least not as per the finalised list of usability requirements in Table 48). This information for all the possible defined and non-defined R values is illustrated in the Venn diagram in Figure 37 below. The dotted line separates the two sets of R values for the two T1-T3 and T2-T3 comparisons.

Figure 37: All defined and non-defined R values for T1 to T3



R values for the T1-T3 comparison

In the T1-T3 comparison, the actual number of requirements in R1 and R2 are not so important because as mentioned earlier, the first set of requirements are focused exclusively on ensuring reliability and efficiency (as well as satisfying basic functionality). The efficiency of operations is considered to be moderately effective by the users, so no major attention was paid to those aspects. There was no attention apparently on other important usability aspects either, namely learnability, rememberability, and user satisfaction, which suggests there may have been a narrow understanding of what usability entails or that the concept of usability itself was either not properly understood or given enough importance during the conceptualisation process as well as during the process of development. At the same time however, the new list of requirements is focused exclusively on these other aforementioned usability aspects. However, this seems to be due to satisfaction with the existing functionality and reliability of the software generally, as was expressed by the teachers during the meeting. In other words, the exam system is considered to be working well as it was (prior to the UgCD trial), so the meeting was used as an opportunity to mainly find ways of working to enhance the system in these other aspects.

R values for the T2-T3 comparison

Similarly, in the T2-T3 comparison, the high figure of 10 for R1 suggests UgCD also enabled a better capture of requirements in comparison with the period prior to the UgCD implementation for improving the usability of the software. Notably, the developers were relying on usability questionnaires for feedback previously whereas UgCD provided a complete methodology for improving usability. It is also to be noted that R2 = 1 and R3 = 5, so with one exception, all five other requirements capturing during T2 were recaptured during T3. The exception was the imprecisely worded reliability requirement of students being “able to undertake the exam without any problems”. Under the focus provided by UgCD, the meeting participants were able to not only be precise in identifying sound problems to occur the most, but also to come up with the idea of making the software perform a check to make sure the sound is working before letting students start the exam. So again, the R2 figure is of no consequence here either. As for R3 being 5, this was expected as the developers already had in mind to make those changes.

10.4.3 Completeness and preciseness ratings

Table 51 presents the completeness and preciseness ratings for each usability aspect or requirement, and for each of the three occasions (T1 to T3), which are then compared in the subsequent subsections. The completeness values were also given in Table 49, and the preciseness ratings are given in the final column for each requirement.

Table 51: Completeness and preciseness ratings of the usability requirements (T1-T3)

Aspect	**Specific requirement**	**T1** (original capture)	**T2** (Pre-UgCD)	**T3** (Under UgCD)	**Completeness rating**	**Preciseness rating**
Usability Requirements						
Reliability	Students should be able to undertake the exam without any problems.	✓	✓		T1: 1 T2: 1 T3: 1	3
	Perform sound check to make sure sound is working prior to starting exam.			✓		1
Learnability	Highlight important elements using colour.			✓	T1: 0 T2: 3 T3: 7	2
	Highlight important elements using bold text.		✓	✓		2
	Increase font size of main heading.		✓	✓		1
	Use another clearer and larger font.			✓		1
	The help system (for teachers and also students) should be made context-aware			✓		1

Aspect	**Specific requirement**	**T1** (original capture)	**T2** (Pre-UgCD)	**T3** (Under UgCD)	**Complete-ness rating**	**Precise-ness rating**
	instead of presenting the list of contents.					
	Develop a simple user documentation for students so they can be prepared before the exam.		✓	✓		1
	Make log in and log out screens in both English and Arabic			✓		1
Remembera bility	Use icons to represent elements for ease of identification.			✓	T1: 0 T2: 1 T3: 3	1
	Represent the time remaining information using a timer.			✓		1
	Display photo of student		✓	✓		1
Efficiency in Use	Provide navigation system for moving to the next question.	✓			T1: 1 T2: 1 T3: 3	1
	Allow students to quickly jump to another question without having to go through the back/next buttons.		✓	✓		1
	Allow the keyboard navigation and tab keys to be used by students to quickly navigate the interface without having to rely on the mouse.			✓		1
	Reduce the number of clicks required in the teacher's interface (currently 5) to get to a particular student's previous exam record.			✓		1
User Satisfaction	Make interface more colourful using different colours.			✓	T1: 0 T2: 0 T3: 1	1
Total		2	6	15	T1: 2 T2: 6 T3: 15	T1: 4 T2: 9 T3: 17

10.4.4 Indications of completeness

A key determining factor for completeness is the measure for R1 relative to R2.

Completeness of requirements captured during T3 relative to T1

In the T1-T3 comparison, the fact that R1 is much greater than R2 (15 compared with 2) indicates that as far as completeness is concerned, UgCD has definitely enabled a better capture of usability requirements. The C values for the old (C1) and new (C2) completeness rating totals happened to correspond exactly with R2 and R1 respectively in this case due to the mutual exclusivity of the captured requirements. The completeness ratings for each aspect of usability were given in Table 51 above. The improvement in this

case was more than seven-fold (factor is 7.5 or 750%).

$$R1 = R2 \times 7.5 \text{ (specifically, } C2 = C1 \times 7.5) \qquad [T1\text{-}T2]$$

The fact that learnability, rememberability and user satisfaction were not considered previously suggests that the initial list of requirements was lacking. And, the importance attached to these aspects and to the whole concept of usability itself during the UgCD trial suggests that the new set of requirements are more complete in terms of their inclusion of usability aspects, and coverage through specifications of requirements. The UgCD trial therefore made the developers think more widely about usability in terms of not only basic technical aspects of designing software, but also aspects related to the cognitive domain (learnability and rememberability) and aesthetical appeal (user satisfaction). And, the new list can be regarded as more comprehensive and complete despite this being an improvement to existing software instead of a recreation.

Completeness of requirements captured during T3 relative to T2

The second comparison was to measure how much more completely UgCD was able to capture requirements compared to the previous state when usability questionnaires had begun to be used. The T2-T3 comparison also confirms the superiority of UgCD, as the increase in the number of captured requirements was from 6 to 15, of which 5 (R3) were captured on both occasions. The one requirement (R2) that was captured without UgCD, but not under UgCD is a vague requirement (Ensure students can take exams without any problems) whereas UgCD helped to be more precise about the likely situations. UgCD helped to capture 2.5 times (or 250%) more usability requirements than the previous method while the attempt was being made to improve the interface, so the formula in this case is:

$$C2 = C1 \times 2.5 \qquad [T2\text{-}T3]$$

10.4.5 Indications of preciseness

As for preciseness of the captured usability requirements, the expert opinion was that the new list of requirements were framed precisely. The main expert who made the ratings was the leading developer of the software development team who also consulted his other team members before presenting the ratings. That is, all the requirements were expressed clearly and without any ambiguity. As per the ratings defined in 6.7.2 Adopted measures, which range from 1 (very explicit) to 4 (too vague), all the new requirements were rated as either

1 (explicit) or 2 (reasonably clear), so none of them were rated as 3 (not so clear) or 4 (too vague). The rating given for each requirement specification is stated in the final column of Table 51 above. Those that were rated 2 were the first two items under the learnability aspect: 'Highlight important elements using...' It was suggested to specify exactly which elements are considered to be important that should be highlighted.

Preciseness of requirements captured during T3 relative to T1

Of the old requirements, the description given for the reliability aspect was considered to be not clear, as the problems were not specified, and was therefore given a rating of 3. Without being able to compare this the old set of requirements thoroughly, and the fact that there are only two old usability requirements to compare with, it cannot be claimed for certain that UgCD has enabled the requirements to be expressed more precisely. Nonetheless, an estimation was made of the two measures of preciseness pertaining to the old (P1) and new (P2) set of requirements.

T1-P1: Total 4 for the 2 captured requirements

P2: Total 17 for the 15 captured requirements

It is to be noted that a lower rating indicates a higher degree of preciseness, and vice versa. If the total rating (which ranges from 1 to 4) is divided by the total number of captured requirements, then the range of possible values for preciseness is from 1 (most precise) to 4 (least precise). On this scale, the value for P1 is 2 and for P2 is 1.13, so P2 is less than P1 (P2 < P1), so P2 has a higher preciseness rating than P1. This indicates that, in terms of preciseness, there has been an improvement. The exact extent of improvement is 43% (to the nearest percentage) calculated as follows:

Gain from T1-P1 = 2 to P2 = 1.13 = (2-1.13) / 2 = 0.433 (or 43%) [T1-T3]

Preciseness of requirements captured during T3 relative to T2

For the T2-T3 comparison, it is apparent that the difference in preciseness is smaller as five of the six requirements were mutually captured and have the same preciseness ratings. However, the vagueness of the first reliability requirement (when UgCD was not applied) shows that UgCD helped to be more precise and to avoid being imprecise. The P values in this case are:

T2-P1: Total 9 for the 6 captured requirements (value of 1.5)

P2: Total 17 for the 15 captured requirements (value of 1.13)

The improvement from P1 to P2 in this case, is 24% to the nearest percentage calculated as follows:

Gain from T2-P1 = 1.5 to P2 = 1.13 = (1.5-1.13) / 1.5 = 0.244 (or 24%) [T2-T3]

10.4.6 Completeness and preciseness of specific usability aspects

Table 52 summarises the totals of the completeness and preciseness ratings and the P values for preciseness, for each usability aspect. For the T1-T3 comparisons, the totals for the completeness ratings show that C2 (15) > C1 (2), suggesting a 7.5 fold increase in the level of completeness in terms of usability. Even in the second (T2-T3) comparison, there was a 2.5 fold increase in the measure of completeness. The above figures also confirm the observation that the improvement was most pronounced in terms of learnability (there being 7 captured learnability related requirements compared with 0 originally and 3 previously). The T2-T3 comparison showed that some reliability, learnability, rememberability and efficiency requirements were able to captured without UgCD, but UgCD excelled particularly in helping to capture requirements more directly related to user satisfaction.

The preciseness ratings for individual aspects show that the P2 values range between 1 and 1.29 (with the overall being 1.13), and that these are highest for reliability, rememberability. Efficiency, and user satisfaction. This compares with the range of 1 to 3 for T1-P1 with an overall value of 2, and the same range for T2-P1 but with a lower overall value of 1.5. The greatest improvements in preciseness appear to have been made for the first two aspects, namely reliability and learnability.

Table 52: Completeness and preciseness ratings for each usability aspect (T1-T3)

Aspect	**Number of requirements or Completeness rating totals**			**Preciseness rating totals**			**P values*** (total preciseness / completeness)		
	T1	**T2**	**T3 (**UgCD**)**	**T1**	**T2**	**T3** (UgCD)	**T1**	**T2**	**T3** (UgCD)
Reliability	1	1	1	3	3	1	3	3	1
Learnability	0	3	7	0	4	9	-	1.33	1.29
Rememberability	0	1	3	0	1	3	-	1	1
Efficiency	1	1	3	1	1	3	1	1	1
User satisfaction	0	0	1	0	0	1	-	-	1
Overall total or P value	2 (T1C1)	6 (T2C1)	15 (T3C2)	4 (for T1)	9 (for T2)	17 (for T3)	2 (T1P1)	1.5 (T2P1)	1.13 (T3P2)

**Lower is better (closer to 1).*

10.5 Summary of the UgCD trial

A limited implementation of UgCD was made at one higher education institution that only involved teachers as the software users (so not students), did not allow for the researcher to be physically present, and was restricted to only a single meeting. The researcher could not therefore be actively involved during the UgCD trial, which was due to cultural restrictions, and which itself was limited in the way that UgCD was applied given that the implementation was confined to a single meeting. In terms of the proposed UgCD framework, this means that only the first three stages for gathering requirements were able to be carried out, and there were no subsequent meetings for any further consultations to refine the requirements. Also, the requirements were captured during this meeting for improving an existing piece of software rather than for creating a new one or for re-creating it.

Despite these aforementioned restrictions and limitations, the developers were guided as to how to apply UgCD and focus on improving usability, especially with respect to conducting the meeting. The software was an online exam system being used at KAU, and requirements were captured for improving the usability of its interface. The main users of the software are students who use it to take their exams online and teachers who check the progress of their students. The flow of tasks is illustrated in Figure 34. The users, their roles and tasks were therefore successfully identified.

Notably, whereas previously the requirements that were gathered pertained to functionality,

reliability and efficiency, under UgCD the attendees were able to come up with several new requirements relating to learnability and user satisfaction. UgCD therefore helped those present in the meeting to focus on usability concerns in a wider context that included aspects related to learnability, efficiency and user satisfaction. This was in contrast to the focus when the initial requirements were previously captured, which were narrowly concerned with only providing the necessary functionality and ensuring reliability (Table 47, Table 48). Notably, the implementation of UgCD enabled a 7.5-fold increase in terms of completeness and 43% improvement in terms of preciseness compared with the original capture at the time of construction, and a 2.5 gain in completeness and 24% improvement in preciseness compared to the attempt to make improvements prior to UgCD using usability questionnaires.

The newly captured requirements were therefore more complete than before as they covered aspects that had not been previously considered. The quality of preciseness was also satisfactory but the lack of previous data (only 2 usability requirements captured previously) for comparison did not enable stating with certainty whether UgCD enabled an improvement in this area or not.

Chapter Eleven

Post-UgCD Trial

Chapter 11: Post-UgCD trial

11.1 Introduction

This chapter reports on the third phase of the study, i.e. the post-UgCD trial phase following the demonstration of UgCD at the selected institution. It reports on the software changes implemented, especially in terms of the changes made to the interface, the suggestions given, and the developers' own future development plans. It also presents the results of the post-UgCD trial survey with the meeting participants, and the findings of the interview with the leading developer. The suggestions were given for making further improvements in the usability of the interface. An image of the old interface is shown in Figure 38, and of the new interface in Figure 39 and Figure 40.

11.2 Software changes implemented

As a result of the meeting, some changes to the software were made, but not all suggestions that were made at the meeting were taken on board by the developers. The reasons were not made clear, but it seemed to be due to time constraints, priorities of other projects, and managerial restrictions. Nonetheless, the developers acknowledged that the UgCD methodology helped them to be more focused on the usability aspects and thereby bring about certain improvements to make the interface more usable, especially in terms of make it easier to use and a little more appealing. A developer remarked that knowledge of UgCD helped to raise greater awareness of usability in a more comprehensive sense and more thoroughly than before, and that its implementation had a big impact through making all who were present at the meeting focus intensely on usability and gathering requirements around the idea of helping users to accomplish essential tasks. Also, that it had a much greater impact for improving the interface than their previous attempt, which in comparison lacked the kind of clarity and focus that UgCD enabled them to have.

The EMES online exam system in use at the institution has 3 interfaces for students. The first interface is the log in screen, the second is the main exam screen, and the third is the end of exam screen. These screens appear as follows:

- First screen – log in screen, which appears when the student is ready to log in.
- Second screen – main exam screen, which appears during the course of the exam.

- Third screen – end of exam screen, which appears when the exam has ended and the user has logged out.

An image of the old interface of the main screen, as used for the student side of the exam system, is shown in Figure 38, and of the improved version of the interface in Figure 40 together with an overlay of English translations of its key elements. The two interfaces appear to be similar of course in many ways, but whereas some changes are obvious, such as the change of header, and the increased font size of the main heading 'Main Screen for Student Test', others are finer changes that also made quite a difference. In particular, it can be seen that the title is larger and prominently displayed, a different font is now used for the question and answer areas, which have also been made a little larger, and some coloured areas have been used to improve the clarity of the new interface. In addition, it can be seen that there is an improved area for 'linked questions' on the right side. This was in response to the requirement specified as “Allow students to quickly jump to another question without having to go through the back/next buttons.” It serves two important functions, which are to enable the student user to see the current question in the context of the whole set of questions, and secondly to quickly jump to another question non-linearly, i.e. without having to go through them sequentially.

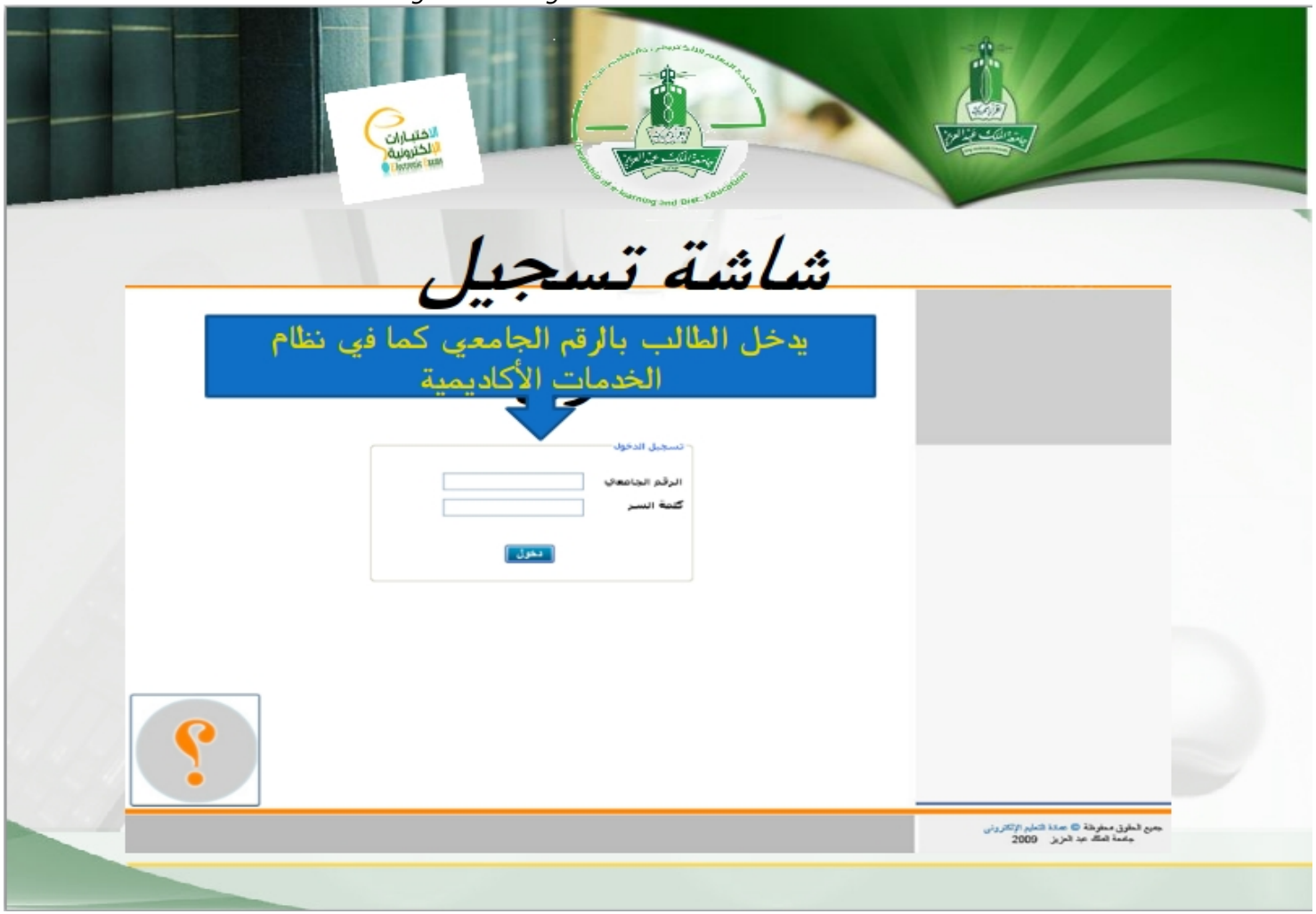
Another major change made is the inclusion of the student's photo underneath the particulars. This is displayed on all pages when the student takes the exam. The developers claim it helps both the students and invigilators to see who is taking the exam, and makes it easier to ensure that the right students is taking the right exam. Also for students, the requested password feature was implemented that now enables them to change to another computer desk if needed to continue taking the exam in case of any technical problems.

For teachers, two important functionality changes were made that are not apparent on the interfaces below. These relate to providing teachers with some automatically generated feedback not only on individual students, such as to identify their possible weaknesses, but also a general picture by aggregating responses. In addition, teachers are now able to put a mark directly in the exam software, and they can specify the day on which a certain exam has to be taken, and students are not able to change the day.

Figure 38: The old interface of the EMES online exam system

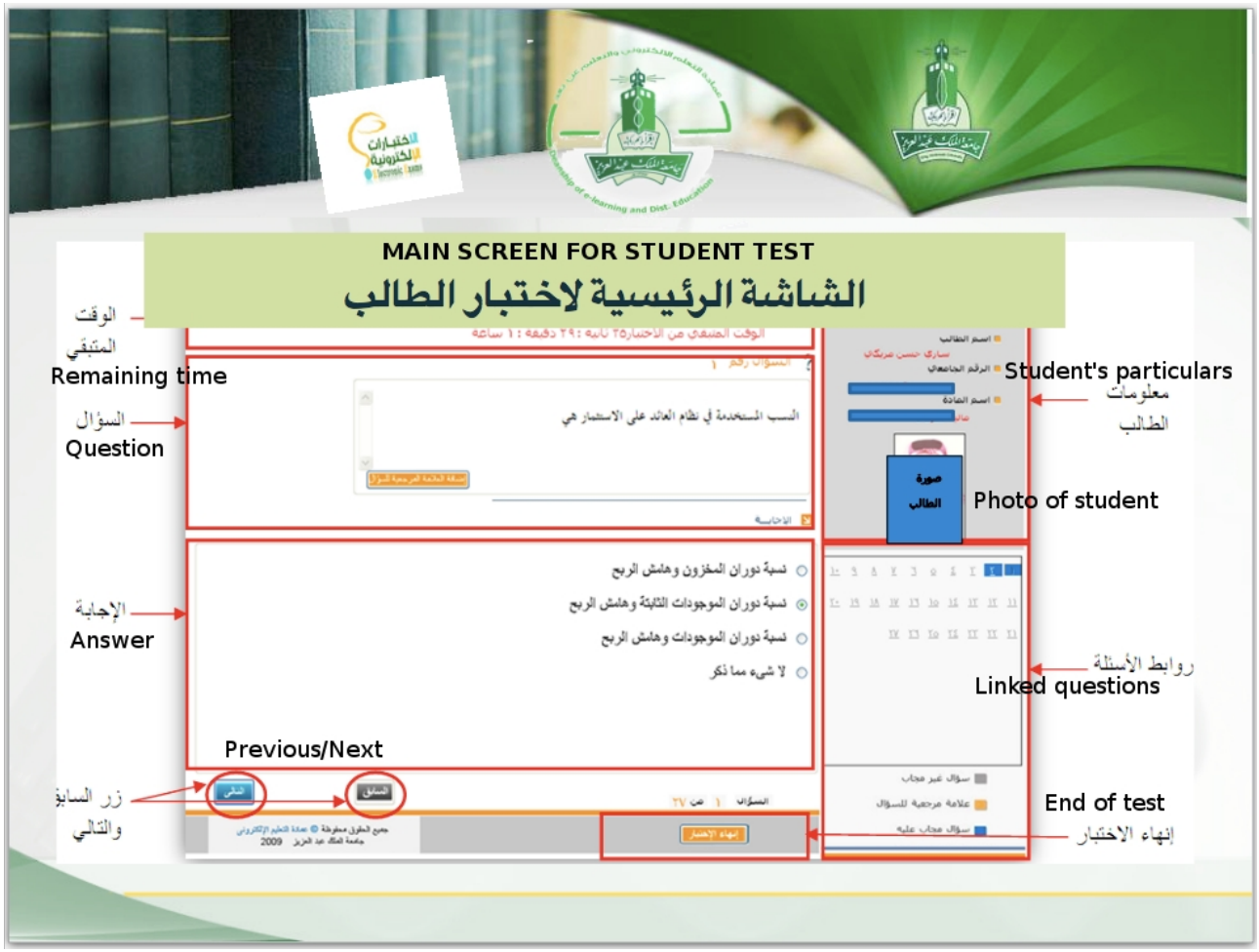
Source: Image provided by KAU

Figure 39: Log in screen of the new interface



Source: Image provided by KAU

Figure 40: The new interface of the EMES online exam system



Source: Image provided by KAU (adapted)

11.3 Suggestions and future development plans

11.3.1 Suggestions made for further improvements in usability

The new interface and exam system still fell short of what could have been a more ideal outcome in the opinion of the researcher and based on the requirements captured during the UgCD trial. Besides incorporating changes taking into account the remainder usability requirements, the following two additional suggestions were made for further usability enhancements:

1. It was suggested the developers should definitely consider designing some icons to further improve visual clarity. The back and next buttons for instance, still relied on text labels whereas arrows could be used as well. Moreover, the current buttons do not look attractive.

2. It was suggested a timer should be visible on the interface to clearly show the time instead of relying on a simple textual display of the information.

The possibility for addressing the first item was made, but the icons used for illustration were not original[4]. They merely serve to show the type of icons that would be possible to use to not only enhance the look of the interface, but more importantly, to make it more user-friendly. These would be especially useful for new students taking their exams online for the first time, but for all students generally too given that during an exam, students would want to do things as quickly as possible. These icons were shown to the developers, but they said any icons would have to be designed from scratch and it was not a priority at the moment. They did not rule out the possibility however, of including such icons in the exam interface in a later revision.

4 Icons taken from: http://www.menu-icons.com/glossy-menu-icons/index.htm

Table 53: Sample of icons that could be used to enhance the interface further

Information	Student	Question	Next	Previous
				
Start	Stop	Correct	Wrong	Time remaining
				

11.3.2 Future development plans for the exam system

In addition to considering the above-mentioned recommendations for improving usability, the developers also assured they may work on incorporating the remaining requirements captured during the UgCD trial during future development. In particular, they were interested to incorporate or at least begin to work on implementing the following five items in the near future:

- To enable students to also conduct essay type exams, as they are currently only able to do true or false or multiple choice type questions.
- Incorporate more automatic checking, marking, and aggregate analysis options to provide useful data for teachers and administrators.
- Providing a context-sensitive help system instead of just presenting a list of contents.
- Improving efficiency by reducing the number of clicks required to perform certain tasks that were made clearer using UgCD.
- To focus more on improving the efficiency and reliability of the system relative to adding more functionality, as the coding has become very large and complex.

11.4 Post-UgCD trial survey and interview

11.4.1 Survey of meeting participants

A short and quick survey was held among the participants of the meeting during which UgCD was demonstrated for recapturing requirements. A total of 14 of the 16 invited meeting participants took part in the survey. Five of these were developers and nine were teaching staff. This survey was conducted after the trial and after the software changes described above were made and the further suggestions were offered. The purpose was to gather the views or perceptions of the participants about their experience of the UgCD trial in terms of its effectiveness for capturing usability requirements. This was followed by a final interview with the leading developer.

The survey participants were asked to indicate their extent of agreement with ten statements about their experience of the UgCD trial and their views or perception of its effectiveness with respect to software usability. The items were chosen based on what is claimed about UgCD and the experience of the demonstration. The first statement was on UgCD's ability to help focus on usability, the second two on its ability to help identify user roles and tasks. The next five were related to the five usability aspects, and the last two were given with a view to seeing how well UgCD may be accepted based on the two important factors in the Technology Acceptance Model (TAM), namely perceived usefulness and perceived ease of use.

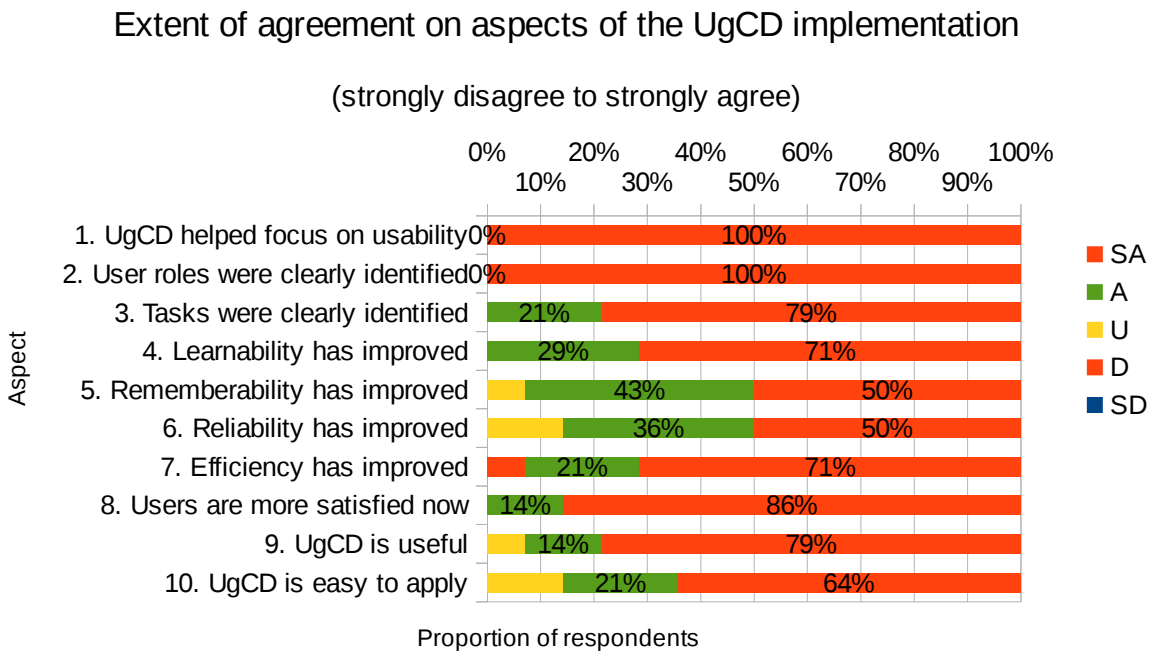
A 5-point Likert scale was used for respondents to indicate the degree of their agreement on a scale ranging from 'strongly disagree' to 'strongly agree' with a neutral 'unsure' response option in between. A copy of the questionnaire is attached in Appendix N: The post-UgCD survey and interview questionnaire, and the number of responses for each degree of response and the overall mean values are given in Table 54 below.

Table 54: Breakdown and mean values of the post-UgCD survey responses

No.	Aspect	SD	D	U	A	SA	Mean
	Value >	1	2	3	4	5	-
1	UgCD helped to focus on usability.	0	0	0	0	14	14
2	The roles of the users were clearly identified.	0	0	0	0	14	14
3	The tasks for each user role were clearly identified.	0	0	0	3	11	13.4
4	The learnability of the software has been improved.	0	0	0	4	10	13.2
5	The rememberability of the software has been improved.	0	0	1	6	7	12.4
6	The reliability of the software has been improved.	0	0	2	5	7	12.2
7	The efficiency of the software has been improved.	0	1	0	3	10	12.8
8	Users are now more satisfied with the software.	0	0	0	2	12	13.6
9	UgCD is a useful methodology.	0	0	1	2	11	13.2
10	The UgCD methodology was easy to apply.	0	0	2	3	9	12.6

The results presented in the table above show that the proportion of responses for all the ten items is greatest for the fifth degree of response, which is a positive indication of 'strongly agree' with the statement. This is particularly high for the first two items, which state that (1) UgCD helped to focus on usability, and (2) The roles of the users were clearly identified. In these two cases, the mean values are the maximum value of 14, as all 100% of the respondents indicated they strongly agree. Agreements with the remainder items are also high with the exceptions of statements 5 and 6 for which the overall extent of agreement is noticeably a bit lower. This is apparent, for instance, from the half (50%) 'strongly agree' indications and the higher proportion of 'agree' indications. The relative proportions of indications for each response option and for each statement can be seen more clearly in Figure 41 below.

Figure 41: Results of the post-UgCD survey



An ordering of the mean values, presented in the chart in Figure 42 below, confirms the picture presented above with respect to the most (1 and 2) and least (5 and 6) agreed with statements. The relative ordering of the others is as indicated. These results show that, as perceived by the meeting participants, UgCD excels most at helping to focus on usability, and in helping to identify user roles and tasks. Of the five usability aspects (indicated in items 4-8), user satisfaction is likely to be greatest followed by improvements in terms of learnability and efficiency. The other two aspects, namely rememberability and reliability, are perceived as having been least improved. Finally, although the usefulness of UgCD is perceived to be high, for which the mean values is 13.2, the overall agreement with the tenth statement, that UgCD is easy to apply, is low with a mean value of 12.6.

Figure 42: Post-UgCD survey responses ordered by mean value

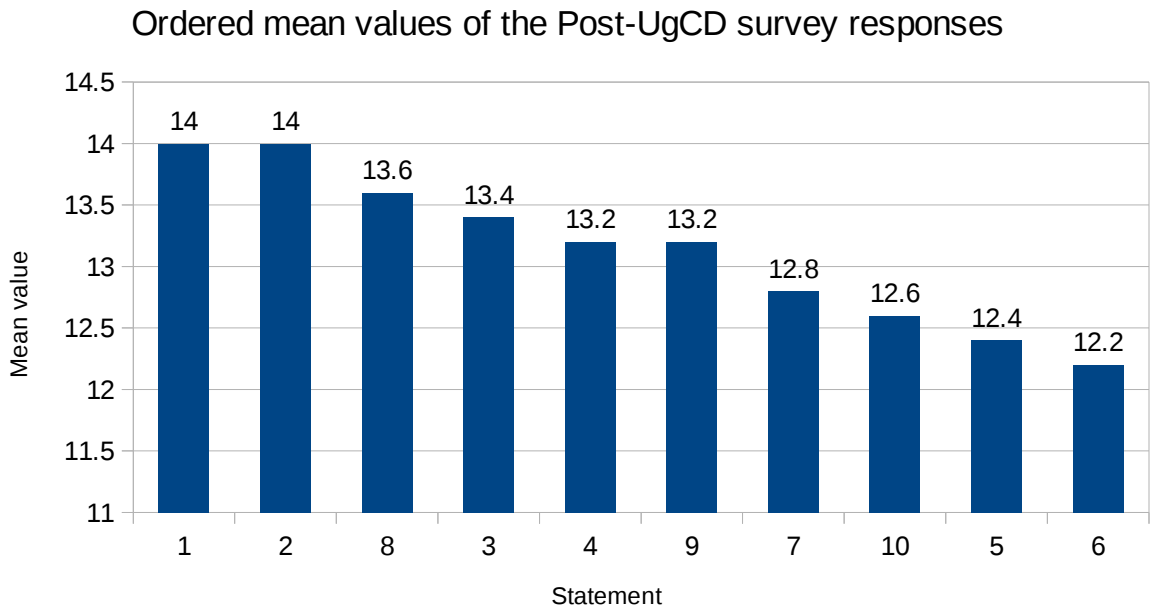
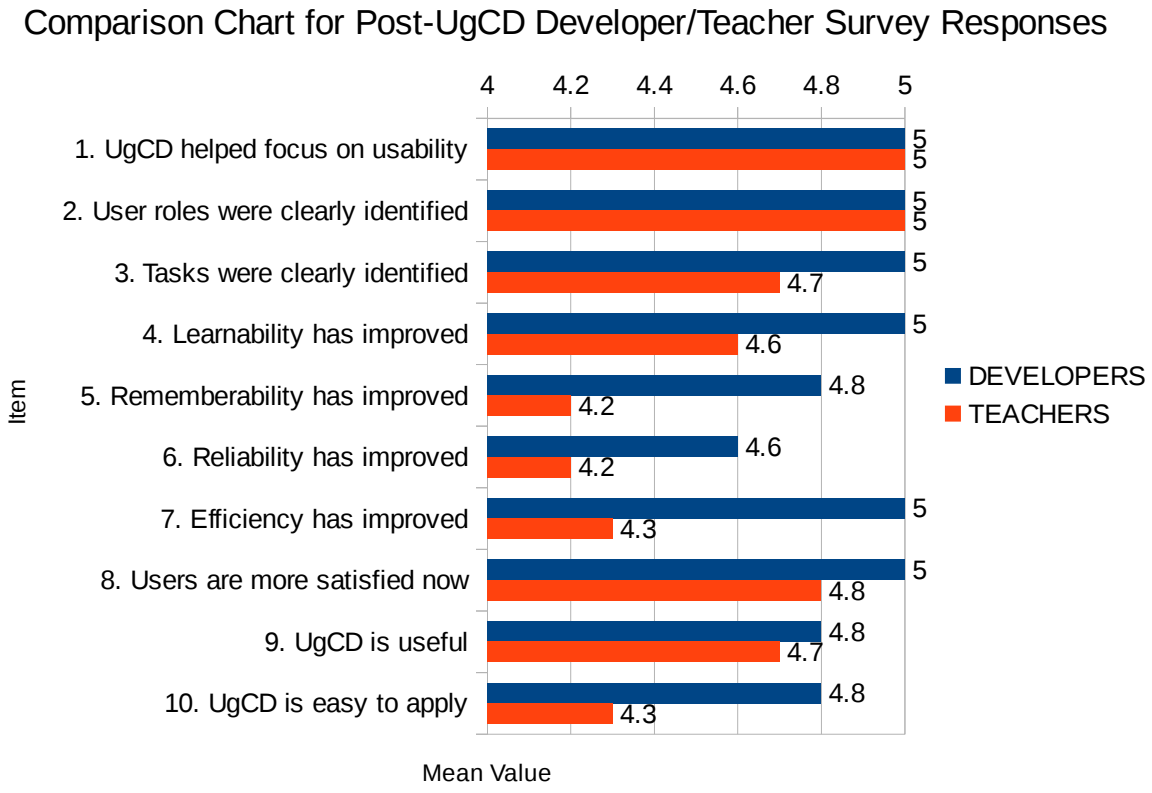


Figure 43 below presents a chart of mean values for developer and teacher responses separately and over the same range 0-5 of mean values (4-5 showing) for ease of comparison. The raw data for the two categories of respondents is attached Appendix N: The post-UgCD survey and interview questionnaire. The data shows there are some similarities as well as marked differences between the two categories of respondents. The responses for items 1 and 2 bear the greatest similarities, as all of the respondents indicated the same extent of agreement, in this case strongly agreed. The greatest differences are for items 5, 6, 7 and 10. For all these items, the developers indicated a greater extent of agreement whereas the teachers as a whole indicate a markedly lower extent of agreement. These items pertain to improvements having been made in terms of rememberability, reliability and efficiency, and UgCD being easy to apply. This suggests that overall, the developers were relatively more satisfied with the implementation whereas the teachers present during the trial were relatively less satisfied. From the raw data, it can be seen that the majority of both categories of respondents strongly agreed with all of the ten items. There were only at most a couple of respondents who gave responses to some items that were less than the extent of strong agreement in the case of developers, and agreement in the case of teachers.

Figure 43: Post-UgCD Survey Responses - Comparison Chart



11.4.2 Interview with the leading developer

The interview with the leading developer was held after the implementation of changes to the software, and conduction of the Post-UgCD trial survey. It was arranged to inquire into his personal views of the experience of the UgCD demonstration, and his perception of UgCD for capturing usability requirements completely and precisely in his professional capacity. The developer was asked to comment on five open questions relating to UgCD's ability to help focus on usability, to help improve usability in terms of the five chosen usability aspects, its strengths and weaknesses, its ability to help capture requirements completely and precisely, and the likelihood of adopting UgCD. The precise wording used is given in Appendix N: The post-UgCD survey and interview questionnaire, and the replies given are attached in Appendix O: Post-UgCD interview questions and their replies.

The replies indicate a positive perception of the trial and UgCD, and the leading developer seems to be especially enthusiastic about the role of UgCD in terms of efficiency. On helping to focus on usability, the reply was affirmative that “UgCD helped very much”,

and this was mentioned as its “biggest strength”, and it also helped the team be “very clear” on the tasks and finding ways to facilitate their accomplishment efficiently. UgCD also helped to think about other aspects besides efficiency, such as learnability, and to appreciate the importance of user satisfaction as well.

When asked about the capability of UgCD to help capture usability requirements completely and precisely, the reply was again affirmative, that it is “very good” for this purpose, as UgCD enabled the developers to come up with “a comprehensive list”, and the focus also helped them to “be precise” about what was wanted or needed. On the likelihood of using UgCD for future development projects, the developer indicated “yes, we can use [the] UgCD process for being clear on the tasks especially, and then thinking of usability aspects”. This reiterates that UgCD was perceived to be especially useful in these respects. Importantly, when prompted for whether they would involve and consult with users including students, he acknowledge the useful contribution made by teachers, and suggested involving students in this same way can be done in order to “take good advantage of this... so we can make more efficient software”. Again, it was clear that efficiency is the main concern, and although they already solicit views of students using SUS, it is possible they may involve users more after their positive experience with UgCD.

11.5 Summary

The software changes implemented as a result of the meeting are detailed in this chapter, and can be seen in the images of the new interface, and further suggestions were given for making more improvements in usability. These improvements made to the interface as an outcome of the UgCD trial are shown in Figure 39 and Figure 40, and the additional suggestions that were made to improve the usability even further related to the following three aspectual areas:

(1) Learnability aspects, such as by applying colour, highlighting and employing icons for making it easier for users to learn to use and to use quickly;

(2) Readability aspects, such as by using a more suitable font and font size;

(3) Attractiveness aspects, such as by using colour for distinguishing certain parts.

This chapter also reported on the outcome of the post-UgCD trial survey and interview, which were held with 14 of the meeting participants and one leading developer respectively. The survey responses show the greatest agreement is with the capability of

UgCD to help focus on usability and identify user roles and tasks. Improvements in usability were perceived mostly in terms of learnability and efficiency, but the overall usefulness of UgCD is perceived as being high. The interview data confirmed UgCD's capability to help focus on usability, and highlighted the importance attached especially to improving usability in terms of the efficiency of the software. It is also likely UgCD may be used by the development team in their future software development projects.

Chapter Twelve

Concluding Discussion and Recommendations

Chapter 12: Concluding discussion and recommendations

12.1 Introduction

This chapter present an analysis of the key points from the results and findings during the three stages of the primary research (survey, interviews and trial), and synthesises them to point out the overall key findings of the study, which are then discussed in detail for each phase. The study aims, objectives are revisited to show the extent to which they have been achieved. The discussion then delves on a critical evaluation of the theoretical and methodological implications of the study, including the research instruments. Recommendations then made as per the aim to develop a framework for guiding developers in implementing UgCD for gathering usability requirements as well as general recommendations as an outcome of the research. Finally, the chapter points out the key limitations and delimitations, and makes recommendations for further research in this area.

This research compared and contrasted plan-driven and agile methods with a view to justifying the need to adopt a balance of both types of methods in software design. It then examined User-centred Design (UrCD) approaches in the context of usability research, and introduced the concept of Usage-centered Design (UgCD) based on the works of Constantine and others. Key differences between UrCD and UgCD were highlighted in Table 4. Essentially, UgCD incorporates a model-driven phase into an overall agile framework whilst being focused throughout on the user tasks (usage) and ensuring high levels of usability.

As pointed out in 2.6.2 Issues with pure agile methods, both pure agile and plan-driven methods, and even in UrCD (1.3.1 Key features), there tends to be an undue focus on user-centric aspects and a loss of attention on usage, i.e. the actual important tasks. It was also noted there is a lack of research on software usability, especially in the field of e-learning (Srivastava et al., 2009) despite its importance for didactics and pedagogy (Novak et al., 2010), and that the way in which requirements are currently handled tends to be inadequate (Kavitha & Thomas, 2011).

In order to show how UgCD excels in addressing usability issues over methods that claim to be user-centered, which are usually based on pure agile methods, UgCD was applied for capturing usability requirements. Therefore, the areas of usability engineering and requirements engineering were also explored in depth, the latter of which is currently

receiving increasing interest (Zhang, 2010). Also, UgCD was detailed as to how the methodology works, and specifically how it it can be used for helping developers and users focus on usability and gather usability requirements. Usability was understood to be comprised of the five aspects of learnability, rememberability, reliability, efficiency, and user satisfaction, and the captured requirements were tested with respect to their extent of completeness and preciseness. These five aspects are those identified by Constantine (2011) as being most important for the e-learning context (4.7.1 Usability aspects for e-learning software). And, the two quality characteristics of completeness and correctness (which was treated as preciseness in this study) are identified by the IEEE (5.4.2 Requirements gathering recommendations and issues) as important software requirement specifications (SRS).

12.2 Discussion of key results and findings

12.2.1 Discussion of pre-UgCD survey results and interview findings

Summaries of the survey results and interview findings are presented at the end of the chapters in 8.6 Summary of survey results and 9.3 Summary of interview findings respectively. The survey solicited the views of 212 software developers working in higher education institutions throughout the Saudi kingdom, and the interviews enabled 20 of them to provide further insight into their conceptions and practices relating to usability with respect to its different aspects.

The finding from the survey that requirements for facilitating users in tasks is not thought to be captured well is a significant one because it shows the potential for UgCD, which is specially designed to facilitate users in accomplishing tasks. Also, UgCD relies heavily on frequent meetings and continuous consultations, so again, the finding that meetings (B9) are not used so much is significant and further indicative of potential for promoting UgCD among Saudi developers. Similary, the results for part C on usability testing methods show that UgCD can be usefully applied by these developers given its focus on ensuring the software developed is highly usable.

The correlation analysis on the survey data further revealed the importance of the first two variables associated with usability, namely minimal learning time (A1) and ease of use (A2). These were correlated the most with the other variables, and hence appear to be associated the most with usability among the sample surveyed. Of the capturing methods

for usability requirements, the factor of use of secondary research (B6) was correlated the most, and hence the prominent variable. These results helped to identify the most prominent factors and methods in relation to usability, and gave insight into the thinking on usability among the developers surveyed. The prominence of A1 (minimal learning time) in particular shows that it is the dominant component of usability. Given the large proportion of variance that can be explained with this component, it therefore best represented the variation in the multivariate data set. In other words, the first two variables (A1 and A2), especially the first (A1), and to a lesser extent the third (A3), were found to characterise the concept of usability in the view of the survey respondents.

The interview findings highlight a lack of reliance on methods of observation (while the software is being used), and especially the holding of meetings for consultations with users (Q1). They also confirm the narrow understanding of the concept of usability, which does not seem to encompass engagement and other aspects related to user satisfaction, and learnability (Q2). Rather, usability is mostly understood as pertaining only to establishing ease-of-use and ensuring reliability. Consequently, there is an over-reliance on detailed planning (Q4,5 8), lack of post-development testing of usability (Q7). Also notably, it was apparent during the interviews that developers are content with their existing procedures (Q9, 10), and even if they wanted to try something else, they are generally restricted by their management. As for UgCD, the interviews confirmed the methodology was unheard of (Q11) although an interest was expressed by a few (15%) developers to learn about it.

12.2.2 Discussion of the UgCD trial

In spite of the limited manner in which the UgCD trial was conducted, as mentioned in 10.2.4 Context of the trial, it was apparent that the single meeting helped the attendees to focus on usability as understood in a wide sense; as encompassing learnability and user satisfaction. It aided them in gathering usability requirements for these aforementioned and other aspects of usability that had been neglected earlier and which led to making some improvements to the software. These improvements are detailed in 11.2 Software changes implemented and exhibited in Figure 40. Notably, the implementation of UgCD enabled a 7.5-fold (750%) increase in terms of completeness and 43% improvement in terms of preciseness compared with the original capture at the time of construction, and a 2.5 (250%) gain in completeness and 24% improvement in preciseness compared to the attempt to make improvements prior to UgCD using usability questionnaires.

Importantly, this shows that there was an inadequate attention to usability previously whereas this was not the case for functional requirements. In fact, the previous set of requirements were focused almost exclusively on providing functionality, and although some new functional requirements were identified for teachers during the UgCD trial, no more functional requirements were identified for students. This may be because of a general satisfaction with the student side of the interface, or due to the fact that students were not present at the meeting.

The fact that the UgCD trial was undertaken in a restricted way impacted the outcome of the recapture experiment by not allowing the captured requirements to be refined or new ones to be identified during further meetings and consultations with users. In particular, it was mentioned that further face-to-face meetings play a very important role in UgCD, and the fact that these did not take place not only limited the trial, but could also have affected its validity. For instance, the few requirements that were not so precisely defined, i.e. which did not have a P value of 1, may have been worded more precisely later. At the same time, the further meetings and consultations in UgCD allow for changed and new requirements to be taken into consideration. However, as pointed out by Constantine et al. (2003), although refinement and improvement through further meetings and consultations, as well as through repetitive cycles of user testing and feedback, take place in UgCD, they "are not the driving forces in the design process" as in UrCD. Instead, there is relatively greater emphasis on the initial design and modelling. Also, as the researcher was not present, it is not clear how well the UgCD methodology was applied generally during the main meeting. In spite of these restrictions however, the first three stages of the framework seem to have been applied sufficiently enough to have at least demonstrated how users are involved and requirements are captured under UgCD.

As regards the constructs of completeness and preciseness, it was apparent that these qualities were lacking when the requirements were first captured, and from the change in focus that these are issues. This is in spite of the fact that the actual base line of the initial set of requirements was no longer available. The validity of this comparison (T1-T3) may be questioned on this basis, but the differences were still stark, and besides, the second comparison (T2-T3) also indicated the possible superiority of UgCD in capturing usability requirements. Also, it assumes of course that completeness is understood to mean covering all the identified aspects of usability, that is, in a wider sense rather than a narrow sense as only pertaining to reliability and efficiency. Consequently, there was a noticeable lacking in

attention to especially more cognitive and aesthetic aspects of software design, and to features that could facilitate learning to use the software (learnability). As noted in finding a way to deal with a possible sound issue (by making the system do a performance check), the same is the case for preciseness.

There is a possibility that the increases in the completeness and preciseness of requirements may partially be related to the fact that in the requirements recapturing experiment, the user participants were already familiar with the software and its shortcomings, had the opportunity to evaluate the implemented software. Given also that the intent to improve usability specifically was made evident, it could be that even by applying another methodology other than UgCD would have yielded similar results. However, the UgCD methodology provided a systematic and methodical means to achieve this goal, which also ensured for instance, that the interactions were productive, that all participants participated, and that no leakage of requirements crept in (12.6.1 Framework and procedure). Moreover, the UgCD implementation made all the trial participants focused on collaboratively developing an initial model to work with, which is an element that is missing from incremental agile methods wherein greater importance is attached to seeking feedback from users instead.

The fact that refinement of the requirements (through further meetings, consultations, testing and feedback) did not take place does not mean that this latter stage is unnecessary, since improvements were still made irrespectively, as more improvements could have been made. Nor does it mean that the implementation cannot be transferred to an agile environment with which many developers in the Saudi kingdom were found to be unaccustomed generally. The results support the observation that UgCD helped to capture usability requirements more effectively, the usefulness of collaborating with users, and the potential of UgCD for incorporating a valuable plan-driven phase in agile frameworks.

Improving learnability further should be considered to be very important for a timed exam system so that students can concentrate on actually undertaking the exam rather than wasting valuable time trying to figure out how to use the system. Similarly, the second is important for improving visual clarity and is also related to improving the learnability aspect. The third area was considered important to further enhance the visual appeal of the interface, which still looked rather dull despite some improvements.

Overall, the UgCD trial was a productive experience because it helped not only to raise a

little awareness of this methodology, but also in that it promoted the importance of usability in software design in a wider sense of the term than seems to currently prevail. As for preciseness of the usability requirements captured during the trial, the expert opinion was that they were framed precisely, i.e. clearly expressed without any ambiguity although a recommendation was given to be more precise about which elements are described as 'important' for which highlighting was requested. But, as there was no overlap in the two (previous and new) sets of requirements (Table 48), so it was not possible to see whether the degree of preciseness had actually improved, stayed the same or deteriorated. Nonetheless, the precision in the requirements was probably made possible due to the focus on usability and the first three stages of the framework must have helped in that regard.

12.2.3 Post-UgCD trial survey and interview

The post-UgCD survey results suggest that UgCD lives up to its claim of helping to focus intensely on usability as well as to be clear on the user roles and tasks. In particular, this helped the developers to find ways of enhancing the efficiency of carrying out those tasks. This corroborates the findings from the study of Patton (2003), which also showed UgCD to especially help improve software efficiency, as was mentioned in 4.1.2 Potential of UgCD in improving software quality. Also, the leading developer acknowledged the capability of UgCD to provide “a comprehensive list” of usability requirements, and that the usability focus also helped them in being precise about the requirements. As a result of the experience, the perceived usefulness of UgCD is high, and it is therefore likely that, at least to some extent, UgCD will be applied in future software development projects. On the other hand, the survey result for low perceived ease of use, and the developer response to being asked about its possible weaknesses, confirm the earlier finding of UgCD perhaps being seen as involving a lot of preparation and user involvement. However, the greater capability of UgCD to capture usability requirements has been established, so it is possible that it could encourage some changes in practices by being more user centric, consulting with them and involving them in the process. This confirms the need to realise that capturing requirements is very much a social process rather than merely a technical one that can be undertaken by the developers alone (Macaulay, 1993). Cooperation with users has long been advocated in software design, as this aforementioned reference is more than two decades old, so it is about time their needs, views, perceptions and requirements are taken seriously.

12.3 Extended discussion of key results and findings in light of the literature review

The most important results and findings from both the survey and interviews and on the basis of the outcome of the UgCD trial were as follows:

- Holding meetings with users is not popular among Saudi developers, as found in both the survey and the interviews.
- Usability is mostly about ensuring ease-of-use, reliability and following established standards rather than learnability and engagement.
- Greater awareness is needed, not only about UgCD specifically, but also about the importance of usability in software design and ways of enhancing usability, and in capturing requirements appropriately for this purpose.

Notably, it seems that the following two of the issues identified by Macaulay (1996) persist in the context of the software development field in Saudi Arabia:

- Lack of communication between developers and users, especially with students;
- Lack of awareness of the importance of usability in its comprehensive understanding, and knowledge of UgCD;

Also, the gathering of usability requirements tends to be focused primarily on functional aspects and ensuring reliability rather than satisfying more user-centred needs related to aesthetics and overall satisfaction, which can also impact adversely on ease of use and learnability if neglected.

12.3.1 Meetings

UgCD shares with JAD the use of structured and facilitated meetings as an important means for developers and users to get together both for identifying users, roles and tasks, and for gathering requirements (Table 10). Meetings are arranged to encourage user involvement, facilitate communication between the two groups (Coughlan & Macredie, 2002), and to apply dynamic group techniques to help refine ideas (5.5.3 Joint Application Design). Under UgCD specifically, the meetings are held frequently so they are central to the UgCD methodology, and importantly, these frequent meetings help to deal with any omissions, irregularities and ambiguities about the requirements (5.6.1 Outline of process).

Also, the focus on prioritising usability by modelling task cases increases the likelihood of developing highly usable software (Juristo et al., 2003).

If Saudi developers are to gain from the potential benefits of focusing on usability aspects of software design, and specifically from the benefits of the UgCD methodology, then it is paramount for them to become accustomed to communicating with the potential/existing users of their software, and especially to hold frequent meetings with them. UgCD relies heavily on such frequent meetings, so it would be imperative for this kind of focused meetings to be accepted as the norm if UgCD or any other meeting reliant methodology is to have any chance of being adopted for software development.

A single main meeting, as was conducted during the UgCD trial in this study, may be adequate to get started on a software project and maybe define the users, roles, tasks and some essential requirements, but continuous consultations through frequent meetings provides additional benefits. In UgCD, these additional meetings help to obtain feedback and refine the process including the gathered requirements. The single meeting therefore helped to initiate and develop the requirements dialogue, but the UgCD trial did not benefit from allowing to make any further refinements to the gathered requirements using JITR.

Moreover, UgCD expects these meetings to be conducted face-to-face and in a conversational manner in order to facilitate collaboration (Constantine, 2011). In fact, as described in 5.6.2 Detailed description of process, UgCD expects significant user involvement and involves a highly collaborative process for gathering requirements, and this is what distinguishes UgCD from UrCD. Given these expectations of UgCD, it seems it would be very difficult to implement it in a highly conservative Saudi society. This is also consistent with the expectations ascertained from the cultural analysis in 1.7.1 Cultural context. For implementing UgCD, it may be necessary therefore to make a whole raft of associated changes to the way software is developed. If face-to-face meetings are difficult to arrange, especially for mixed gatherings, then live online conferencing may be used as an alternative, but extra care may need to be taken to ensure the atmosphere facilitates open conversation rather than becoming inquisitive, and to encourage attendees to work collaboratively.

12.3.2 Concept of usability

The second main finding was that the concept of usability seems to be understood in a very

narrow and limited sense as only concerning basic functionality and ensuring reliability and efficiency at most. It was not common to find developers perceive usability in a wider sense as also including such aspects as those related to learnability, rememberability, and ensuring user satisfaction.

This is not entirely unusual though. As pointed out in 4.2 Introduction to usability, various definitions of usability have been offered, and there has been no consistent understanding of what actually constitutes usability among software developers. It appears that Saudi developers generally, have an understanding of usability that is more in line with the definition provided by ISO/DIS 9241-11, which identifies effectiveness, efficiency and satisfaction, but makes no mention of ease of use or capability for being learned. Satisfaction may be understood vaguely here, so usability may be understood as being simply about making software effective and efficient, which would include the notion of reliability. UgCD could help developers take more interest in the cognitive (learnability and rememberability) and aesthetic (user satisfaction) aspects.

It is not the case that UgCD ignores functionality; rather, determining functionality is the starting point in UgCD as it is defined by use cases (Gulliksen & Boivie, 2001) (4.1.2 Potential of UgCD in improving software quality). Also, identifying roles and their inter-relationships may help guide decision relating to functionality that can be incorporated in the design (Ferreira et al., 2005: 7). What UgCD does is to then make the team of developers and users think beyond mere functionality, reliability, efficiency, etc. to focus on enhancing usability in terms of learnability and user satisfaction as well. This leads to the third point and appreciating the importance of usability, as understood in its more comprehensive sense and incorporating all five of the usability aspects that were selected in this study.

This narrow understanding of usability also reflects the observation (noted in 4.2 Introduction to usability) of there being no consistent understanding within the software development community of what actually constitutes usability, and that aspects related to more aesthetical concerns and user satisfaction are more common only in newer definitions. It may therefore be the case that these relatively newer conceptualisations of usability have not yet established themselves amongst developers in the Saudi kingdom. At the same time, the importance of user satisfaction was apparent among the users themselves. Thus, there may also be a cultural dimension to the differences in awareness, as well as differences in the conceptualisation of usability between different types of

stakeholders.

12.3.3 Importance of usability

The importance of usability has been stressed in several places in this paper, especially in 4.3 The importance of usability. For instance, Novak et al. (2010) consider it to be very important, especially for e-learning, as it helps to develop better systems with improved didactical and pedagogical approaches. This study highlights the need for promoting the importance of usability among software developers in Saudi higher education institutions, especially those having worked on developing e-learning software. This would be necessary before UgCD specifically can be promoted as a way of enhancing software usability, or else UgCD can be used as the medium itself for promoting usability. If the importance of usability is accepted, only then would it be possible for the concept of usability to be properly understood; for the whole range of usability aspects to be accepted as being essential for ensuring usability comprehensively, and for also appreciating and adopting the practice of developers holding frequent consultations and meetings with existing and potential users. The three key points are therefore inter-related, but the importance of usability would need to be accepted first.

12.4 Revisiting of aims, objectives, research questions and hypotheses

12.4.1 Aims and objectives

This section revisits the aims and objectives in light of the above results and findings. The aim of this study was to develop a framework for guiding developers in capturing usability requirements more completely and precisely than they would otherwise. The main research objective was 'to develop a framework of principles for achieving improved capture of usability requirements through UgCD in terms of completeness and preciseness'. To help achieve this objective, the study ascertained how usability requirements for developing software are captured at present by surveying and interviewing software developers in higher education institutions in Saudi Arabia, and how well the existing requirements capturing process is satisfactorily able to capture usability requirements. A UgCD trial then showed how UgCD can be used to improve the usability requirements capturing process for enhancing software usability by demonstrating the process during the development of an e-learning software.

12.4.2 Research questions and hypotheses

The following two hypotheses were formed in this study (1.5 Research questions and hypotheses): The process for capturing requirements under UgCD yields a more complete (H1) and precise (H2) list of requirements necessary for enhancing software usability in terms of learnability, rememberability, efficiency and reliability in use, and user satisfaction. The precise wording for each of the two is stated in 1.5 Research questions and hypotheses as well as the research questions. The solution for the main research task of finding a suitable framework to guide developers in capturing usability requirements completely and precisely is presented in 12.6.1 Framework and procedure below. The sub-research questions involved the tasks of:

(2) Finding out how usability requirements are captured at present,

(3) Finding out how well the existing process at a selected institution is able to capture usability requirements, and

(4) Establishing whether UgCD can enable to capture these more completely and precisely.

Ascertaining how usability requirements are captured at present for developing software in higher education institutions in Saudi Arabia was successfully done by means of the survey and interviews described earlier. Both of these phases of the research provided detailed and insightful information on the understanding of usability among these Saudi developers, how well they think their existing processes capture requirements for each of the specific selected usability aspects, the methods they currently use for ensuring and testing for usability, and so on. In this regard, the research led to gaining useful information about the *current* position to answer the second research question.

The capability of the existing process to capture usability requirements was ascertained during the UgCD trial from the data gathered for the two time periods defined as T1 and especially T2, the latter of which involved the use of the SUS usability testing questionnaire. As per the data summarised in Table 52, T1 was only able to provide capture of 2 usability requirements, and T2 of 6 usability requirements. Also, the degree of preciseness for T1 was determined to be 2, and for T2, a value of 1.5 was obtained. This information answers the third research question.

The UgCD trial led to being able to evaluate the completeness and preciseness of the

requirements that were re-captured during the T3 period, despite the restricted implementation. In contrast to the above values for T1 and T2, T3, during which UgCD was implemented, resulted in capturing 15 usability requirements, and the preciseness improved with a value of 1.2 (being closer to 1 indicating most precise wording). This information satisfies the fourth research question.

It was clear that there was a shift in focus from the previous (pre-UgCD) state to the new state (under an implementation of UgCD) from functional requirements to more aesthetic requirements, although a more detailed quantitative comparison of the two sets of requirements could have been made. Cultural and time restrictions only allowed for a single meeting to be conducted and without the participation of the researcher although under her guidelines for implementing UgCD and focusing on usability. This prevented stages 4 and 5 identified in Table 19 to be undertaken adequately, which would have involved usability testing and further refinement of the captured requirements. Only the first three stages were carried out extensively, which involved (1) role/task identification, (2) task modelling, and (3) requirements capturing. The UgCD trial thus highlighted the issues and obstacles in the way of realising the ultimate goal of promoting usability.

Nonetheless, it was apparent that previously (without UgCD), there was a limited understanding of usability and a focus only on identifying and satisfying basic functional requirements whereas the UgCD trial, albeit in a restricted and limited manner, helped to focus more on usability (in a wider sense as discussed above). The quality of preciseness of the gathered usability requirements was satisfactory, but whether this was an improvement from before was uncertain, and greater completeness was confirmed from the change in focus. It can be deduced therefore that the first hypothesis pertaining to completeness is upheld whereas the second pertaining to preciseness is uncertain.

12.5 Theoretical and methodological implications and contributions

12.5.1 Theoretical implications

The construct of usability was described and discussed at length in the literature review with the whole of Chapter 4: Usability in software design dedicated to usability specifically, and Chapter 5: Requirements analysis, gathering and capturing to requirements. This ws important given that usability is often neglected in the development of e-learning software (Kruse, 2002). Also, UgCD was introduced in Chapter 3: Usage-

centred software design and development as a software design methodology that specially focuses on enhancing usability, and the way requirements are gathered under UgCD was presented in 5.6 Requirements gathering under UgCD. As noted in 1.3.2 UgCD and usability, this study has probably pioneered an investigation of the role of UgCD specifically during the phase of gathering requirements.

This study promotes the UgCD methodology on the basis that it encourages a strong focus on usability. Also, given the predominance of user-centred methodologies (UrCD), it was shown that, for instance, selective user involvement and being model-driven can help to better target usability whilst also ensuring all essential users, roles and tasks are identified and that the software can enable the users to fulfil their roles and accomplish those tasks. Key differences between UrCD and UgCD are highlighted in Table 4 and Contrast between User-Centred and Usage-Centred Design.

The inclusion of both modelling as the prime method of design and the insistence on substantial user involvement (Table 12) also highlights the balanced nature of the UgCD methodology. The comparison and contrast of plan-driven and agile development methods in Chapter 2: Plan-driven and agile development methods: a comparative perspective showed that a balanced approach can provide for greater adaptation and flexibility in design whilst also maintaining necessary control to ensure task completion. Other such balanced methods were also examined (2.7 Existing combined agile-plan driven methods), but these usually take the form of adjuncts to an agile method, which makes the combined method lose its coherency. It was also noted that the original agile manifesto, which popularised agile methods, did not even incorporate usability engineering, hence the reliance on these additions to the essentially agile methods (4.4 Usability and other software quality characteristics in agile methods). This could also explain why usability is not given sufficient prominence, although it was pointed out this is also due to difficulties in measuring usability (4.8 Usability testing and measurement).

Again, with the predominant interest nowadays in agile methods and general unawareness of UgCD, this study could have the potential of bringing knowledge of this methodology to the attention of more developers, especially those concerned about usability issues. As an extensive piece of research, this study also adds to what thus far has been only very few studies on the potential of UgCD for enhancing usability (Table 5).

As regards the gathering of usability requirements, this study has examined in depth the

different types of requirements, requirements gathering frameworks, different models for requirements, etc. In particular, it has highlighted the importance of gathering requirements (5.1.2 Importance of gathering requirements), the importance of usability requirements (5.2.3 Usability requirements), and shown how requirements are gathered under UgCD (5.6 Requirements gathering under UgCD). As with usability, this focus on requirements was also important because there are still inadequacies at present in how requirements are handled (Kavitha & Thomas, 2011).

12.5.2 Methodological implications

The survey method proved to be useful, as it enabled the views and knowledge of practices to be gathered of a large number of developers in the Saudi kingdom (212 software developers from higher education institutions). However, with hindsight, the questionnaire instrument used (Appendix D: Survey questions) was short, as it only tested for three pieces of information (usability aspects captured by requirements, requirements capturing methods, and usability testing methods). Given the difficulties during the UgCD implementation, it may have been appropriate, for instance, to also enquire about existing software projects and development methods, involvement of users, use of meetings, and the possibility of demonstrating UgCD. Another institution could have been found in which UgCD could have been demonstrated more thoroughly.

The interview method was used satisfactorily to discuss points and practices relating to usability and requirements gathering (with 20 interviewees). The interview questions (Appendix E: Interview questions) were suitably worded and adequate in extracting the required information. Many interviewees had to be prompted, especially for number 7 when they were asked how they tested for usability 'afterwards', but this was due to this being an uncommon practice rather than a deficiency with the wording. Besides, there were no such issues during the pilot phase.

In measuring the construct of preciseness, a minor issue was the scale (6.7.2 Adopted measures) being defined in a reverse manner to what may be considered as intuitive. That is, the scale was defined from 1 (very explicit) to 4 (too vague) rather than vice versa from 1 (too vague) to 4 (very explicit). It may be argued that this is counter intuitive, as described by Fenton's Representational Theory of Measurement. However, the scale was only applied by a single person (the leading developer at the selected institution for the UgCD trial) who was well aware of the scale being defined in this way. It may have been

more important to define this scale differently if, for instance, it had to be used in a survey or otherwise had to be applied by several developers. Related to this issue was the choice of experts to make the preciseness ratings according to this scale. It may have been possible to have asked independent developers to make the ratings, but the opinions of the development team involved in the UgCD implementation was given importance so that they could see for themselves how much difference UgCD can or cannot make.

The UgCD methodology itself was defined by Constantine. Research was conducted into how this methodology is applied, especially during the requirements gathering phase (5.6 Requirements gathering under UgCD). An attempt was then made to apply this methodology at the selected institution during a period when its developers were recapturing requirements to bring about improvements to their online exam system interface. However, as already pointed out, there were some differences in the way UgCD was actually implemented (Table 45), and this issue has already been discussed above in this chapter. The difficulties were not with the methodology itself, but with the way it was able to be applied; the restrictions faced and the limitations imposed. Suggestions have already been made as to how an adaptation can be made to the fourth stage to allow for continued consultation and further meetings.

This issue highlights the challenges that are sometimes faced in carrying out empirical studies. With hindsight, it would have been appropriate to carry out an experimental risk analysis that may have helped to mitigate the risks earlier. It was known from the researcher's earlier study that plan-driven approaches are prevalent in the Saudi kingdom, and that developers there are largely unaccustomed to agile methods. This explains why the development team were comfortable with the phases that involved planning and detailing, and less so when they were required to adopt the more social aspects of the whole process; when interacting and collaborating with users, and the fact that they omitted the fourth phase of refinement. The cultural restrictions were also known beforehand, but it was not known for sure that the developers would not permit the researcher to be present in a group setting at all.

The original difficulty was actually in finding a development team that was willing to implement UgCD in a higher education setting, so when one was found, the researcher was then compelled to oblige with the way that the team considered appropriate for implementing a new methodology. The sample from which the selected team was drawn was the small sample of 20 developers. It may have been possible to implement UgCD

more faithfully if a willing developer had been sought from the larger sample of 212 survey participants, or if other types of organisations were considered other than higher education institutions.

12.5.3 Practical contributions

The demonstration of UgCD showed how this methodology can be used to deal with a pragmatic concern, that is, for increasing the chances of producing highly usable software. Moreover, it showed how this is possible both effectively and efficiently through implementing a systematic process that is geared towards helping the user accomplish all essential tasks (3.1 Introduction to Usage-centred Design). The UgCD approach that enabled this involved task modelling, in which roles and tasks were clearly identified. In an adaptation of Jacobson's Use Case drive approach, it involves defining essential use cases, which define the essence of a use case and are expressed in terms of user intentions and system responsibilities. As pointed out in 5.6 Requirements gathering under UgCD, increasing the chances of enhancing usability is made possible from the focus on prioritising usability by modelling task cases rather than use cases (Juristo et al., 2003). And, as Constantine pointed out, it focuses on actual needs and key requirements. In 5.6.1 Outline of process, the basic process of modelling in UgCD was detailed as consisting of three distinct phases (Figure 17: Basic process of modelling in UgCD):

- **Task modelling** - Raising questions and identifying potential areas for investigation through constructing provisional user role and task models (based on information available at the time)
- **Model sharing** – Sharing of the model with users to solicit their feedback during a main face-to-face meeting and for conducting a 'collaborative requirements dialogue'
- **Model refinement** - Holding of frequent meetings and consultations to deal with any omissions, irregularities or ambiguities (as in JITR), e.g. through holding sharply focused enquiries, investigations, surveys or making observations.

The above process was described in more detail in 5.6.2 Detailed description of process. For instance, it was pointed out that each task case is examined in turn in order to identify the software capabilities required for accomplishing those tasks, and that the requirements dialogue involves 'mutual exploration and negotiation until a consensus is reached between

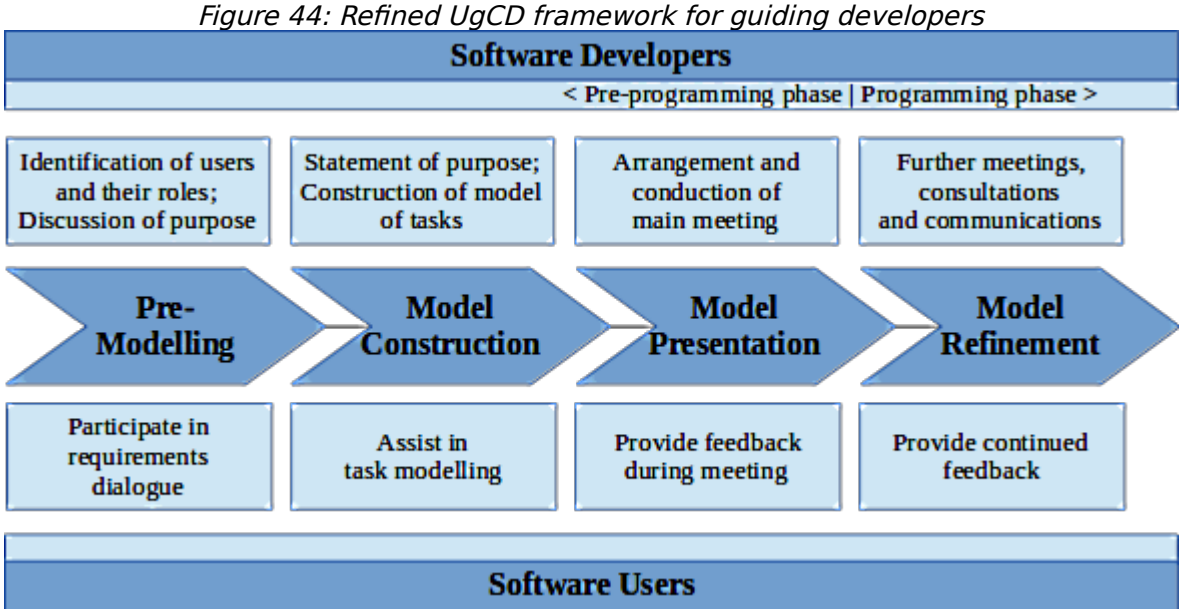
the developers and users'. The role and task models then help to develop abstract prototypes.

Importantly, the implementation of UgCD during the trial was done as per the understanding of the team of developers. The key differences between the instructions given and the actual implementation were highlighted in Table 45. Notably, only the first two phases outlined above were carried out so there was no later refinement of the model. The context in which the trial was done was detailed in 10.2.4 Context of the trial.

12.6 Recommendations and framework for guiding developers

12.6.1 Framework and procedure

Figure 44: Refined UgCD framework for guiding developers



An outline of the UgCD process for capturing requirements was given in 5.6.1 Outline of process and a detailed description in 5.6.2 Detailed description of process. Based on this experience of trying to implement UgCD, and also in light of the survey results and interview findings, the previous tentative framework of the requirements gathering process under UgCD (Figure 23) was accepted as suitable for the purpose without any major modifications. To summarise, it is recommended to implement UgCD by following the procedure outlined below:

1. Select potential participants based on knowledge, commitment and cooperation.

2. Prepare and distribute an agenda in advance of the main meeting to facilitate collaboration.

3. Commence with a discussion or brainstorming session focused on the intended usage or purpose of the system, such as how it is expected to be used, what is expected to be accomplished, and the reason for doing particular tasks.

4. During the collaborative requirements dialogue, gather information, build the model, negotiate, and approve and validate the gathered requirements. This information should include the function (basic capability), form (realisation/appearance of the functions), criteria (desire characteristics), and constraints (on acceptable/possible solutions).

During the meeting it is also necessary to observe the following two points:

- A neutral facilitator should be appointed to oversee the interactions to ensure that everyone present participates.

- The participants should take care to prevent 'creep' and 'leakage' of the gathered requirements, i.e. expanding the system beyond what was initially agreed, and letting additional requirements become part of the system that would give no benefits.

Also importantly, the essential use cases need to be expressed in terms of user intentions and system responsibilities. This is in contrast to how traditional use cases are expressed in the form of user actions and system responses. Specifying an essential use case can be described as being more abstract as opposed to concrete. A table, such as the one shown below, may be devised for this purpose with a separate row for each user intention.

Table 55: Defining an essential use case

User Intention	System Responsibility
Specification of a user intention	Specification of corresponding system responsibilities for each user intention

12.6.2 Recommendations for model refinement

The only recommended modification to the framework is in guiding developers on how to carry out the fourth phase of model refinement, as indicated in the refined UgCD framework presented in Figure 44 above, which is the final outcome of this study. The

researcher stresses that all four phases should be completed for realising the benefits of UgCD. There were no problems in achieving the first three stages of pre-modelling, model construction and model presentation, but for UgCD to be implemented in its complete form and successfully, it is necessary to also adopt the practice of model refinement. This necessarily involves arranging and conducting further consultations and feedback, and for this in turn, as pointed out above, it would be necessary first to make developers appreciate the importance of considering usability aspects in their designs. That is, it is essential to reiterate the need to promote usability; its importance, and an understanding of what usability entails in order for developers to become willing to accept the framework for implementing UgCD.

If despite the acceptance of the importance of usability, there needs to be a transition phase before UgCD can be implemented fully in Saudi Arabia, then it is recommended to find more ways to facilitate the continued meetings and consultations. If further face-to-face meetings cannot be held, then the stakeholders should at least keep in constant contact for open communication so that users can be kept informed of developments and so they can provide valuable feedback. As suggested before, alternative arrangements for meetings can be made, for instance, using web conferencing and live video chat. Continuous consultations are important for refining the requirements, which would likely help to ensure greater perfection in terms of the qualities of completeness and preciseness. The recommendation for the fourth phase in the refined framework therefore allows these alternative methods for facilitating communication and consultation in addition to, or otherwise instead of, holding further meetings. It is also important for developers to involve all key users given that students were not involved in the UgCD trial. Again, if the students cannot be involved in meetings, then continuous consultations can take either the two aforementioned forms, or else students should at least be allowed to provide feedback through a survey or suggestions box.

12.6.3 General recommendations

In summary, based in the results, findings and experience during the trial, the study makes the following recommendations, which can be seen as confirming common observations in the software engineering community:

- Communications between developers and potential/existing end-users should be improved such as by using feedback mechanisms, encouraging openness, and

soliciting responses in surveys. Communication problems in RE were noted by Macaulay (1996), Sutton (2000) and others (see 5.4.2 Requirements gathering recommendations and issues).

- There should be greater receptiveness to holding frequent consultations with users, especially through meetings, which can be promoted by establishing its importance. This problem of not involving users so much was reflected in the responses of several interviewees and confirms the observation for 'hard methods' noted in 2.1 Comparative perspective of agile and plan-driven methods).
- Awareness of the importance of usability must be promoted so that developers would be interested to learn more about it and be receptive to adopting usability-focused approaches to software design. The lack of attention to usability was noted, for instance, by Larman (2002) (see 4.3 The importance of usability)
- Knowledge of the different aspects of usability should be raised so that developers are aware that it also includes cognitive and aesthetic aspects, and not understand it in a limited sense. This confirms of the observation of Rubin & Chisnell (2008) (see 4.2 Introduction to usability).
- UgCD should be demonstrated more widely to show how usability requirements can be captured for potentially enhancing usability. This supports the view of Constantine (Chapter 3: Usage-centred software design and development).
- The prevalence of PMP in Saudi Arabia indicates that most software developers in the kingdom are accustomed to detailed planning before commencing on a project. They would therefore need to be convinced of the benefits of agile methods, especially in the way it can allow to accommodate changing requirements, and the benefits of iterative designs. This may help them to see the need for a balanced approach and make them more receptive to adopting UgCD.

12.7 Limitations, delimitations and recommendation for further research

This study was conducted among software developers engaged in software development in higher education institutions in Saudi Arabia. Apart from developers, only teachers as the software users were involved during the UgCD trial, and the trial was also restricted in that there were no female participants. The researcher was also excluded and for this reason the

implementation of the UgCD trial was also limited compared to what was planned. The 212 survey participants were represented from all over the Saudi kingdom, but the 20 interview participants were only from the cities of Jeddah, Riyadh and Taif. The survey results could therefore be representative of the Saudi kingdom as a whole, but not necessarily the interview findings.

This study has contributed to especially further understanding of the potential role of UgCD for gathering usability requirements for potentially enhancing software usability. This fulfils important gaps in research as UgCD is not currently well-known, research on UgCD has been limited since Constantine introduced it over a decade ago, and research had not previously focused on its potential specifically during requirements gathering. The study has contributed to both fields of Usability Engineering and Requirements Engineering. Further research could be conducted to demonstrate UgCD in several institutions to establish its potential more widely, including outside the Kingdom of Saudi Arabia, for a wider range of software types, and over multiple iterations of requirements gathering or a complete software development lifecycle.

Appendices

Appendix A: Comparison tables

Home ground between agile and plan-driven methods in OSS

Table 56: Home ground between agile and plan-driven methods in OSS

Home-ground area	Agile methods	Open source software	Plan-driven methods
Developers	Agile, knowledgeable, collocated, and collaborative	Geographically distributed, collaborative, knowledgeable and agile teams	Plan-oriented; adequate skills; access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Dedicated, knowledgeable, collaborative, and empowered	Access to knowledge, collaborative, representative, and empowered customers
Requirements	Largely emergent; rapid change	Largely emergent; rapid change, commonly owned, continually evolving – 'never' finalised	Knowable early; largely stable
Architecture	Designed for current requirements	Open, designed for current requirements	Designed for current and forseeable requirements
Refactoring	Inexpensive	Inexpensive	Expensive
Size	Smaller teams and products	Larger dispersed teams and smaller products	Larger teams and products
Primary objective	Rapid value	Challenging problem	High assurance

Source: Boehm (2002) (adapted)

Contrast between User-Centred and Usage-Centred Design

Table 57: Contrast between UrCD and UgCD

User-Centered Design	Usage-Centered Design
Focus is on users: user experience and user satisfaction	Focus is on usage: improved tools supporting task accomplishment
Driven by user input	Driven by models and modeling
Substantial user involvement ● User studies ● Participatory design ● User feedback ● User testing	Selective user involvement ● Explorative modeling ● Model validation ● Usability inspections
Design by iterative prototyping	Design by modeling
Highly varied, informal, or unspecified processes	Systematic, fully specified process
Design by trial-and-error, evolution	Design by engineering

Source: Constantine & Lockwood (2001), p. 4.

Appendix B: Useful features for a Human-Computer Interface (HCI)

Table 58: Useful features for a HCI

Technique	Description
starters	messages that indicate first steps or suggest initial user actions
balloon tips or help	pop-up messages that appear on first encounter or when a user reaches a certain point in a process
screen tips	brief pop-up messages that appear after a delay when the mouse pointer remains over a tool or other object
progressive tool tips	two-level tool tips with an extended message appearing after a second delay; also called "cascading tool tips"
embedded prompts	hints, directions, or explanations embedded within the fields of user interface controls, such as text entry boxes, drop downs, and combo boxes
input prototypes	representative examples of acceptable user input, such as, "Time of day (e.g., 13:30):"
templates	models of the format of acceptable user input, such as, "Expiration date (mm/yyyy):"
start highlighting	graphic or other indication of the place to start within a dialog box or screen
workflow highlighting	a line or other graphic device that visually guides the user through the steps of a process
dynamic affordances and constraints	dynamic changes in appearance to signal allowed and disallowed actions
instructive animation	graphical animation that informs users about what is happening or how to use or interpret something
anticipatory action	automatic actions supplied by the user interface in anticipation of user needs
progressive enabling	the technique of enabling a series of user interface controls in their logical or required sequence
progressive disclosure	the technique of exposing or displaying user interface controls in their logical or required sequence
implicit antecedents	automatic insertion of missing user steps or actions when these are unambiguously implied
idiomatic parallels	using common arrangements, appearance, or hints across different interaction idioms or methods for accomplishing the same task
thematic variation	use of familiar or standard components and interaction idioms in new or non-standard contexts

Source: Constantine & Lockwood (2002)

Appendix C: Research schedule

Original schedule

Table 59: Gantt chart of original research schedule

Month	June	July	Aug.	Sept.	Oct.	Nov.	Dec.	Jan.	Feb.	March	April	May	June	July
Phase	Pre-I			I				II			Post-II			
Refine and complete 15-month report														
Obtain ethics approval for data collection														
Finalise research instruments														
Finalise conference paper														
Conduction of interviews														
Conduction of survey questionnaire														
Analysis of collected data														
Meeting with supervisor/examiner														
Participation in Dubai conference														
Recapturing of requirements using UgCD														
Testing for improved requirements capture														
Write-up of conclusion														
Completion of draft														
Allowance for final adjustments														
Final printing and binding														
Hand-in of final report														

Phase I (June 2013-Nov. 2013): Data gathering and analysis of the present state

Phase II (Dec. 2013-July 2014): UgCD implementation for requirements capturing, and data gathering and analysis of improved state

Phase III: Complete UgCD implementation through to design and development, and usability testing (at the discretion of the participants)

Revised schedule

Table 60: Gantt chart of final revised research schedule

Month/Step	June				July				August				September				October				November			
Shortlist possible institutions for UgCD demonstration																								
Select an institution for the UgCD demonstration																								
Obtain ethics approval to carry out the research																								
Obtain original list of captured requirements (prior to UgCD)																								
Finalise guide for instructing on UgCD implementation																								
Arrange for initial meeting for UgCD implementation																								
Carry out initial meeting as per UgCD guidelines																								
Sort captured requirements from initial meeting																								
Carry out follow-up consultations																								
Analyse all captured requirements																								

Month/Step	June				July				August				September				October				November			
Checking for improved requirements capture																								
Write-up of UgCD implementatio phase																								
Meeting with supervisor and approval for continuation																								
Write-up of conclusion chapter																								
Meetings with supervisor/Final adjustments to paper																								
Printing and binding of report																								
Hand-in of final report																								
Preparation for viva																								
Preparation of conference paper on UgCD implementation																								
Submission of conference paper																								
Attend conference																								
Conduct successful viva																								
Obtain PhD																								

Appendix D: Survey questions

Survey questionnaire for users (teachers and students)

This survey is about your use of the e-learning software at your university.

1. Are you a teacher or a student at the university? ☐ Teacher ☐ Student
2. How often do you use the e-learning software?
   - ☐ Several times a day
   - ☐ Once every day
   - ☐ Several times a week
   - ☐ Once a week
   - ☐ Less frequently than once a week
3. Is teaching/learning the primary motivation behind using the software? ☐ Yes ☐ No

4. Please indicate the extent to which you agree or disagree with each of the items listed in the table below. Students only need to complete parts A and B, and part C is for teachers only.

No.	Aspect	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
	Part A - General questions					
1	The program keeps me informed about what is going on through suitable feedback.					
2	The program keeps me informed about what is going on through prompt feedback.					
3	The terms used by the program are familiar to me.					
4	The information is presented in a natural and logical order.					
5	When I make a mistake, I am easily able to revert back to the previous state.					
6	I feel the program is consistent in its use of words, situations, and actions.					
7	The design is good in that I rarely see an error message.					
8	The range of possible objects, actions and options are clearly visible.					
9	I do not have to keep remembering information as I move from one dialogue to another.					
10	Instructions to use the program are easily available.					
11	The program allows me to perform some frequent actions quicker.					
12	The dialogues only contain relevant and needed information.					

No.	Aspect	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
13	Error messages are expressed in plain language and indicate the problem precisely.					
14	The program gives constructive suggestions in resolving any problems that occur.					
15	Help documentation is easy to find and use.					
	Part B - Further questions					
1	Learning to use the program is easy for first time students.					
2	Icons and other screen elements used are intuitive and self-explanatory.					
3	The range of objects, actions, and options are sufficient.					
4	The interface can easily be personalised.					
5	The layout of the interface is easy to understand.					
6	Navigational objects and tools are clear defined and accessible.					
7	The program provides sufficient guidance to perform tasks.					
8	The wording used in the instructions are appropriate for learners.					
9	The organisation of the content supports learning.					
10	The program makes effective use of media content.					
11	I do not experience any technical problems while using the program.					
12	The program stimulates further inquiry.					
13	The program is interesting and enjoyable to use.					
14	The program provides engaging activities.					
15	The program operates the way it is designed to operate.					
	Part C for teachers only					
1	The program clearly supports students' learning goals, outcomes and objectives					
2	The program allows learning objects to be created easily.					
3	The program allows learning objects to be reused easily.					

No.	Aspect	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
4	The program is able to provide useful evaluative information about learner progress.					
5	The program has sufficient facilities to facilitate teacher-learner interaction.					
6	The program has features that provide motivation to learners.					
7	The program is able to support different learning strategies.					
8	The program clearly supports effective learning.					
9	Abstract concepts are illustrated with specific concrete examples.					
10	Users are provided with sufficient links to external content to support further learning.					

Survey questionnaire for developers

How well do you think the software you have designed in the recent past is able to capture requirements to ensure the following usability aspects?

No.	Aspect	Not well at all	Not so well	Unsure	Reasonably well	Quite Well	Very well
1	Minimal learning time to use the software						
2	Ease of use of the software						
3	Following of established standards in design						
4	Incorporation of metaphors from people's real world experiences						
5	Facilitation of users in successfully accomplishing intended tasks						
6	Minimisation of possible errors						
7	Consistently enabling quick completion of tasks						
8	Reliability of the software to work as expected						
9	Aesthetic appeal of the software						
10	Ability to engage the user						

Which of the following methods do you use to capture requirements to ensure your software is usable? (Check all that may apply.)

- ☐ Field observation
- ☐ Focus groups
- ☐ Interviews
- ☐ Logging details of actual use cases
- ☐ Other: _____

Which of the following methods do you use to check for usability afterwards? (Check all that may apply.)

- ☐ Feature inspection
- ☐ Cognitive walkthrough
- ☐ Expert evaluation
- ☐ Think Aloud approach
- ☐ Heuristic Evaluation
- ☐ Perspective based inspection
- ☐ Other: _____

Appendix E: Interview questions

1. Please describe how you manage to capture user requirements for the software you have developed in the recent past?

2. What do you understand by the term usability?

3. How well do you think your software has been able to ensure usability in terms of how well users are able to learn to use the software?

4. How well do you think your software has been able to ensure usability in terms of how well users are able to use the software effectively?

5. How well do you think your software has been able to ensure usability in terms of how well users are able to to use the software reliably?

6. How well do you think your software has been able to ensure usability in terms of how well users are able to to use the software in an engaging manner?

7. How do you test for usability afterwards?

8. Are you satisfied with your existing procedures to capture software requirements for usability design?

9. Do you think there is scope for improving the way software requirements are captured?

10. Would you be willing to improve this requirements capturing process if it is possible to do so?

11. Are you aware of Usage Centered Design and its possibilities for improving software usability?

Appendix F: Best Practice Document

Best Practice

Title of project: Framework for Improved Capture of Usability Requirements through Usage-Centred Design

School Ethics Committee reference number: ERGO/FoPSE/8613

This is a study on Usage-Centred Design (UgCD) for the development of web-based e-learning software that will be carried out in Saudi Arabia, for the purpose of a PhD dissertation at the School of Electronics and Computer Science, University of Southampton, UK. The focus will be on realising improved requirements capture in order to enhance software usability in terms of learnability, effectiveness, reliability, and capability to engage users.

Existing literature largely relates the adoption of agile methods for web development, and in a User-Centred Design (UrCD) context. UgCD combines elements of both traditional plan-driven and agile methods and is believed to ensure high software usability. In order to demonstrate this, initially, the existing process of requirements capture at selected higher education institutions in Saudi Arabia will be examined through carrying out a survey and conducting interviews. The study will focus only on higher education academic organisations, namely universities, and the study will later be extended to the home institution of Southampton University for an in-depth investigation into requirements capture.

The research will be carried out at two universities in Jeddah, one of which will be King Abdulaziz University (KAU), two universities in Riyadh, and one in Taif. The interviews and survey will complement each other so this will be mixed methods phase of study. A survey questionnaire will be used to gather large scale data among users of e-learning software, and semi-structured interviews will be carried out among key individuals involved in the design and development of e-learning software at the selected institutions. That is, potential participants for the interviews will be solicited from people directly engaged in the development of web-based e-learning software at all the five universities. The purpose of the latter will be to gain insight into the existing requirements capturing processes so as to identify areas for improvement with respect to software usability, and to

later demonstrate the potential of UgCD for enhancing usability. It is anticipated that a minimum of 25 responses will be received. Each potential participant will then be invited to participate by interview meeting, email, or phone, as appropriate.

The survey and interviews will comply with The Data Protection Act. Participation will be anonymous and all required information will be restricted to that which would be relevant for the study. The data collected will not be used for any other purpose. The interview will be recorded using a voice recorder, and notes will be taken of key points made as useful information. The collected data will remain safe and will not be able to be accessed by anyone else except the investigator. The survey data will be aggregated so as not to identify any single participant.

None of the interview questions relate to any personal subject data. All the questions are related only to requirements capture, usability and usage-centred design. Similarly, the survey questions are only related to various aspects of usability and there are no personal or demographic questions. Regardless, the investigator is committed to not use any information to identify any individual and this information will remain unpublished and secure from public access. The answers given by each participant will only be retained for a period of one month after which they will then be deleted completed. Only data summaries will be given and analyses will be made in the PhD thesis report. No data relating to individual subjects will be contained therein that could be used to trace individuals through their responses. Moreover, participants will be able to have access to the published material generated by this study, which will be available by contacting the School of Electronics and Computer Science (ECS) School office at:

> Electronics and Computer Science, University of Southampton, SO17 1BJ, United Kingdom
> Tel: +44 (0)23 8059 6000; Fax: +44 (0)23 8059 6881; Email: school@ecs.soton.ac.uk
>
> Will be published on the ECS eprints websites

Appendix G: Consent Information Document

Consent Information

Title of PhD project: Framework for Improved Capture of Usability Requirements through Usage-Centred Design

School Ethics Committee reference number: ERGO/FoPSE/8613

Each member has the rights to unconditionally withdraw at any time and for any reason, and has the rights to retain any inducement following withdrawal. If the interview is submitted and the participant wishes for the removal of their submission, this can be done by emailing the investigator on:

kja1g09@ecs.soton.ac.uk

or by contacting the School Ethics Committee on:

Electronics and Computer Science
University of Southampton
SO17 1BJ
United Kingdom
Tel: +44 (0)23 8059 6000
Fax: +44 (0)23 8059 6881
Email: school@ecs.soton.ac.uk

The School Ethics Committee reference number should be included in all your communication.

The participant may also refer to the Best Practice document for information related to subject data that is requested as part of the interview. This study does not involve any intrusion that will imply extant risk of unwelcome effects upon a participant's privacy, person, or time. The completion of the interview should not take more than 15-30 minutes.

Appendix H: Ethics Committee Application

Application for Expedited Review

OFFICE USE ONLY Reference number: E ____/____/_____ *[yy/mm/nnn]*

Name of **investigator**(s): **Kholod Jeza Alotaibi**		Date of application: **Signature**(s): *(Please review the definition of 'signature', below.)*	
Name of supervisor(s): **Dr Andrew M Gravell** (if applicable)		**Signature**(s): *(Please review the definition of 'signature', below.)*	
Title of **study**: **Framework for Improved Capture of Usability Requirements through Usage-Centred Design**			
If the **study** is **invasive**, involves **minors**, animals, or coercion, please refer to your Group Representative on the School Ethics Committee.			
		Yes	
1	Does the **study** involve human **participants**? If 'No', approval for the **study** may be dealt with by Chair's Action.	Yes	
2	Is the **risk** of **harm** to any **participant insignificant**?		No
3	Will every **participant** be able to withdraw from the **study** at any time and for any reason?	Yes	
4	Will every **participant** be informed of the true purpose of the **study** before the **study** begins?	Yes	
5	Does the **study** involve deception of any **participant**?		No
6	Does the **study** involve **sensitive data**?		No
7	Is the **study intrusive**?		No
8	Does the **study** involve **subject data**? If 'Yes' or 'Maybe', the **study** requires a **consent document**. Additionally, explain and provide in attached pages the '**best practice**' as it will apply to the **study**. If 'No', the **study** requires **consent information** only. (Note that if the **study** involves **subject data** in respect of **participants** who are ALL students of the University, or does *not* involve **subject data**, the study may be eligible for 'ES' review instead.)		No
9	Does the **study** involve **inducement** to any **participant**? If 'Yes', explain the nature of the **inducement** (eg course credit) in an attached paragraph.		No
Attach a **description** of the **study**. Attach copies of any questionnaires or other **study** instruments. Version number every document.			
Names of ECS SEC		Date of approval:	

reviewing members: (1) (2) (3)	Signatures:

Appendix I: Raw survey data

Table 61: Raw survey data

VAR	LABEL	NW	NSW	U	RW	QW	VW	TOTAL	OWV
A1	Minimal learning time	20	13	17	76	54	32	212	227
A2	Ease of use	0	0	21	42	76	73	212	413
A3	Established standards	4	6	5	44	72	81	212	417
A4	Use of metaphors	18	27	39	55	41	32	212	170
A5	Facilitate users in tasks	19	36	49	28	38	42	212	156
A6	Minimisation of errors	10	17	88	29	33	35	212	163
A7	Consistent/Quick task completion	27	18	28	56	39	44	212	194
A8	Reliability	8	6	12	48	51	87	212	389
A9	Aesthetic appeal	11	14	36	76	28	47	212	237
A10	Ability to engage	19	27	48	41	37	40	212	170
	Total	136	164	343	495	469	513		
	Weight	-2	-1	0	1	2	3		
	TWV	-272	-164	0	495	938	1539		

Table 62: Results for methods used to capture requirements

VAR	LABEL	Q
B1	Field observation	35
B2	Focus groups	157
B3	Interviews	178
B4	Logging details of actual use cases	76
B5	Other -Survey	195
B6	Other -Secondary research	69
B7	Other -Workshops	48
B8	Other -As instructed	71
B9	Other -Meetings	68
B10	Other -Brainstorming	122
B11	Other -None	12
	Total	1031

Table 63: Results for methods used to check for usability afterwards

VAR	LABEL	Q
C1	Feature inspection	58
C2	Cognitive walkthrough	18
C3	Expert evaluation	42
C4	Think aloud approach	9
C5	Heuristic evaluation	5
C6	Perspective based inspection	7
C7	Other -User trial	127
C8	Other -User feedback	91
C9	Other -None	79
	Total	436

Appendix J: PSPP analysis and output

Variable view

Table 64: Variable view of the survey dataset

PSPP_data.sav [DataSet1] — PSPPIRE Data Editor

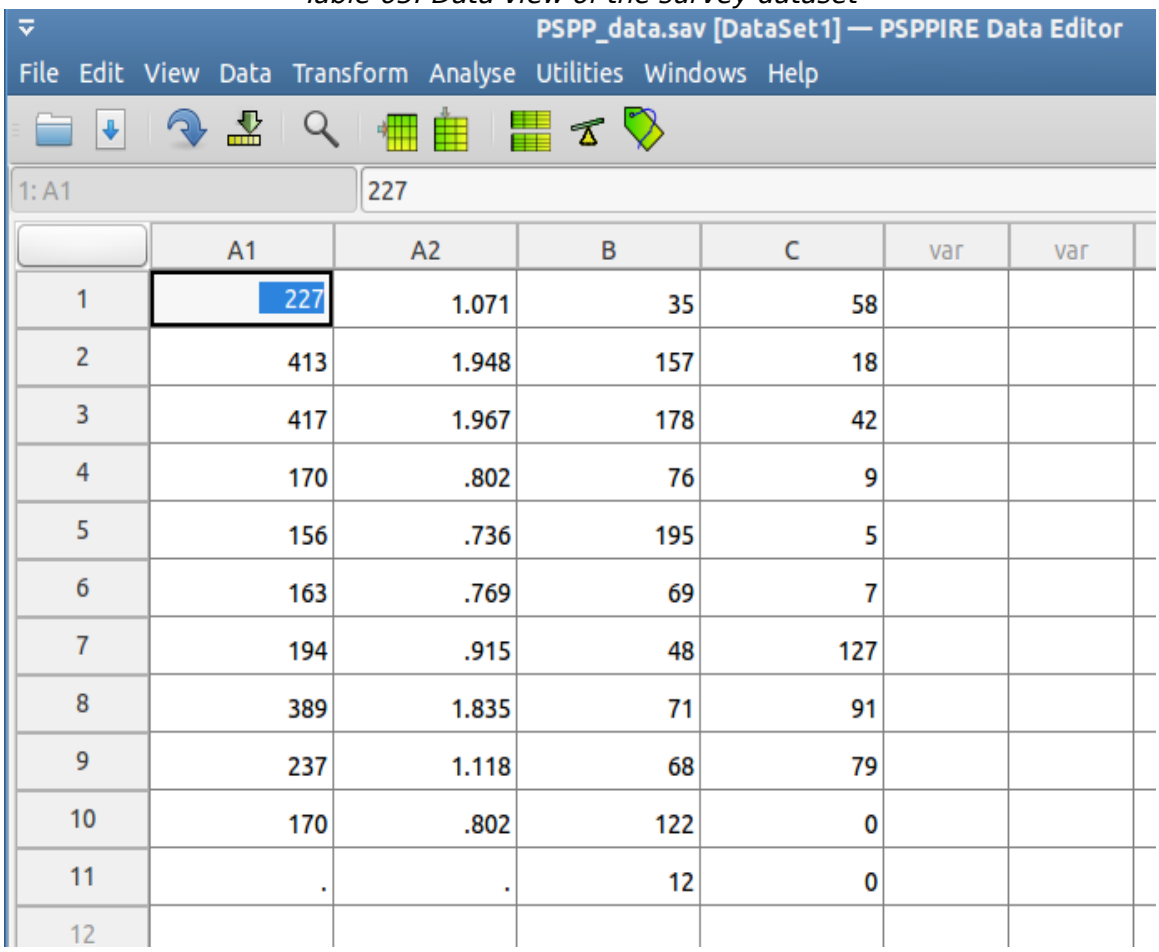
File Edit View Data Transform Analyse Utilities Windows Help

	Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure
1	A1	Numeric	8	0	OWV	None	None	8	Right	Scale
2	A2	Numeric	8	3	Mean	None	None	8	Right	Scale
3	B	Numeric	8	0	Methods	None	None	8	Right	Scale
4	C	Numeric	8	0	Tests	None	None	8	Right	Scale
5										

Where A1 = OWV values and A2 = Mean values obtained for part A of the survey, and B and C are the values obtained in parts B and C of the survey respectively.

Data view

Table 65: Data view of the survey dataset



	A1	A2	B	C	var	var
1	227	1.071	35	58		
2	413	1.948	157	18		
3	417	1.967	178	42		
4	170	.802	76	9		
5	156	.736	195	5		
6	163	.769	69	7		
7	194	.915	48	127		
8	389	1.835	71	91		
9	237	1.118	68	79		
10	170	.802	122	0		
11	.	.	12	0		
12						

Characteristics of OWV values (A1)

OWV

N	Valid	10
	Missing	0
Mean		253.60
Std Dev		108.88
Variance		11854.27
Minimum		156.00
Maximum		417.00

Characteristics of mean values (A2)

Mean

N	Valid	10
	Missing	0
Mean		1.20
Std Dev		.51
Variance		.26

```
|Minimum          | .74|
|Maximum          |1.97|
+-----------------+----+
```

Test for normality of the mean values

Mean

```
+---------------------+-----+
|N          Valid     |   10|
|           Missing   |    1|
|Mean                 | 1.20|
|S.E. Mean            |  .16|
|Mode                 |  .80|
|Std Dev              |  .51|
|Variance             |  .26|
|Kurtosis             |-1.29|
|S.E. Kurt            | 1.33|
|Skewness             |  .84|
|S.E. Skew            |  .69|
|Range                | 1.23|
|Minimum              |  .74|
|Maximum              | 1.97|
|Sum                  |11.96|
|           50 (Median)| .993|
+---------------------+-----+
```

Characteristics of values for recapturing methods (B)

Methods

```
+-----------------+-------+
|N         Valid  |     11|
|          Missing|      0|
|Mean             |  93.73|
|Std Dev          |  60.40|
|Variance         |3648.42|
|Minimum          |  12.00|
|Maximum          | 195.00|
+-----------------+-------+
```

Characteristics of values for testing methods (C)

Tests

```
+-----------------+-------+
|N         Valid  |     11|
|          Missing|      0|
|Mean             |  39.64|
|Std Dev          |  43.56|
|Variance         |1897.65|
|Minimum          |    .00|
|Maximum          | 127.00|
+-----------------+-------+
```

Spearman rank correlation analysis

A2 with B

```
Symmetric measures.
#====================================#=====#==============#========#===========#
#Category          Statistic        #Value|   Asymp. Std.| Approx.|    Approx.#
```

			Error	T	Sig.
Ordinal by Ordinal	Spearman Correlation	.00	.34	.00	
Interval by Interval	Pearson's R	.30	.32	1.37	
N of Valid Cases		10			

A2 with C

Symmetric measures.

Category	Statistic	Value	Asymp. Std. Error	Approx. T	Approx. Sig.
Ordinal by Ordinal	Spearman Correlation	.58	.16	2.86	
Interval by Interval	Pearson's R	.23	.27	1.04	
N of Valid Cases		10			

B with C

Symmetric measures.

Category	Statistic	Value	Asymp. Std. Error	Approx. T	Approx. Sig.
Ordinal by Ordinal	Spearman Correlation	-.23	.31	-1.76	
Interval by Interval	Pearson's R	-.32	.22	-2.47	
N of Valid Cases		11			

Factor analysis

```
---------------------- PART A ----------------------
FACTOR
       /VARIABLES= A1 A2 A3 A4 A5 A6 A7 A8 A9 A10
       /CRITERIA = MINEIGEN (1) ITERATE (25)
       /EXTRACTION =PC
       /METHOD = CORRELATION
       /PLOT = EIGEN
       /PRINT = INITIAL EXTRACTION ROTATION
       /CRITERIA = ITERATE (25)
       /ROTATION = VARIMAX.
```

Communalities

	Initial	Extraction
Minimal learning time	1.00	.94
Ease of use	1.00	.94
Established standards	1.00	.89
Use of metaphors	1.00	.96
Facilitate users in tasks	1.00	.95
Minimisation of errors	1.00	.90
Consistent/Quick task completion	1.00	.96
Reliability	1.00	.90
Aesthetic appeal	1.00	.96

Ability to engage	.96

Total Variance Explained

Component	Initial Eigenvalues			Extraction Sums of Squared Loadings			Rotation Sums of Squared		
	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %	Total	% of Variance	
1	9.35	93.54	93.54	9.35	93.54	93.54	9.35	93.54	
2	.24	2.40	95.94						
3	.11	1.07	97.01						
4	.09	.85	97.86						
5	.06	.62	98.48						
6	.05	.49	98.97						
7	.04	.40	99.37						

Component	Initial Eigenvalues			Extraction Sums of Squared Loadings			Rotation Sums of Squared		
	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %	Total	% of Variance	
8	.03	.32	99.68						
9	.02	.18	99.87						
10	.01	.13	100.00						

Component	Cumulative % of Squared gs
1	93.54
2	
3	
4	
5	
6	
7	
8	
9	
10	

Component Matrix

	Component
	1
Minimal learning time	.97
Ease of use	.97
Established standards	.94

```
#                Use of metaphors#      .98#
#       Facilitate users in tasks#      .97#
#          Minimisation of errors#      .95#
#Consistent/Quick task completion#      .98#
#                     Reliability#      .95#
#                Aesthetic appeal#      .98#
#               Ability to engage#      .98#
#================================#=========#
```

Rotated Component Matrix

```
#================================#=========#
#                                #Component#
#                                #---------#
#                                #    1    #
#--------------------------------#---------#
#           Minimal learning time#      .97#
#                     Ease of use#      .97#
#           Established standards#      .94#
#                Use of metaphors#      .98#
#       Facilitate users in tasks#      .97#
#          Minimisation of errors#      .95#
#Consistent/Quick task completion#      .98#
#                     Reliability#      .95#
#                Aesthetic appeal#      .98#
#               Ability to engage#      .98#
#================================#=========#
```

---------- PART B ----------

FACTOR

```
       /VARIABLES= B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11
       /CRITERIA = MINEIGEN (1) ITERATE (25)
       /EXTRACTION =PC
       /METHOD = CORRELATION
       /PLOT = EIGEN
       /PRINT = INITIAL EXTRACTION ROTATION
       /CRITERIA = ITERATE (25)
       /ROTATION = VARIMAX.
```

Communalities

```
#====================================#=======#==========#
#                                    #Initial|Extraction#
#------------------------------------#-------+----------#
#                  Field observation#   1.00|       .65#
#                       Focus groups#   1.00|       .77#
#                         Interviews#   1.00|       .82#
#Logging details of actual use cases#   1.00|       .86#
#                       Other - Survey#   1.00|       .63#
#         Other - Secondary research#   1.00|       .92#
#                    Other - Workshops#   1.00|       .78#
#              Other - As instructed#   1.00|       .91#
#                   Other - Meetings#   1.00|       .92#
#              Other - Brainstorming#   1.00|       .65#
#                       Other - None#   1.00|       .28#
#====================================#=======#==========#
```

Total Variance Explained

```
#=========#=======================#=======================#===============
#         #   Initial Eigenvalues   #    Extraction Sums of   # Rotation Sums
#         #                         #     Squared Loadings    #         Loadings
#         #-----+--------+----------#-----+--------+----------#-----+--------+
#Component#Total|  % of  |Cumulative#Total|  % of  |Cumulative#Total|  % of  |
```

		Variance	%		Variance	%		Variance
#1	6.22	56.58	56.58	6.22	56.58	56.58	5.33	48.41
#2	1.97	17.94	74.52	1.97	17.94	74.52	2.87	26.11
#3	.99	8.97	83.49					
#4	.60	5.42	88.91					
#5	.45	4.06	92.97					
#6	.31	2.82	95.80					
#7	.19	1.73	97.53					
#8	.15	1.35	98.88					
#9	.09	.81	99.69					
#10	.03	.23	99.92					
#11	.01	.08	100.00					

Component	Cumulative %
#1	48.41
#2	74.52
#3	
#4	
#5	
#6	
#7	
#8	
#9	
#10	
#11	

Component Matrix

	Component	
	1	2
Field observation	.74	.31
Focus groups	.60	-.64
Interviews	.50	-.76
Logging details of actual use cases	.92	.14
Other - Survey	.37	-.71
Other - Secondary research	.94	.20
Other - Workshops	.84	.29
Other - As instructed	.94	.18
Other - Meetings	.93	.20
Other - Brainstorming	.73	-.33
Other - None	.46	.27

Rotated Component Matrix

	Component	
	1	2

```
#                  Field observation# .80|  .06#
#                       Focus groups# .24|  .85#
#                         Interviews# .10|  .90#
#Logging details of actual use cases# .88|  .30#
#                      Other - Survey# .01|  .80#
#         Other - Secondary research# .92|  .25#
#                    Other - Workshops# .88|  .13#
#              Other - As instructed# .92|  .27#
#                   Other - Meetings# .92|  .25#
#              Other - Brainstorming# .50|  .63#
#                       Other - None# .53| -.03#
#====================================#====#=====#
```

---------- PART C ----------

```
FACTOR
       /VARIABLES= C1 C2 C3 C4 C5 C6 C7 C8 C9
       /CRITERIA = MINEIGEN (1) ITERATE (25)
       /EXTRACTION =PC
       /METHOD = CORRELATION
       /PLOT = EIGEN
       /PRINT = INITIAL EXTRACTION
       /CRITERIA = ITERATE (0)
       /ROTATION = NOROTATE.
```

Communalities

```
#============================#=======#==========#
#                            #Initial|Extraction#
#----------------------------#-------+----------#
#          Feature inspection#   1.00|       .79#
#       Cognitive walkthrough#   1.00|       .66#
#           Expert evaluation#   1.00|       .68#
#        Think aloud approach#   1.00|       .87#
#        Heuristic evaluation#   1.00|       .78#
#Perspective based inspection#   1.00|       .90#
#          Other - User trial#   1.00|       .58#
#       Other - User feedback#   1.00|       .84#
#                Other - None#   1.00|       .87#
#============================#=======#==========#
```

Total Variance Explained

```
#=========#=================================#=================================#
#         #      Initial Eigenvalues      #Extraction Sums of Squared Loadings#
#         #-----+------------+------------#--------+------------+-------------#
#Component#Total|    % of    |Cumulative %#  Total |    % of    | Cumulative %#
#         #     |  Variance  |            #        |  Variance  |             #
#---------#-----+------------+------------#--------+------------+-------------#
#1        # 4.79|       53.18|       53.18#    4.79|       53.18|        53.18#
#2        # 2.19|       24.32|       77.50#    2.19|       24.32|        77.50#
#3        #  .73|        8.14|       85.64#        |            |             #
#4        #  .43|        4.76|       90.40#        |            |             #
#5        #  .31|        3.43|       93.83#        |            |             #
#6        #  .22|        2.42|       96.25#        |            |             #
#7        #  .15|        1.70|       97.95#        |            |             #
#8        #  .09|        1.04|       98.99#        |            |             #
#9        #  .09|        1.01|      100.00#        |            |             #
#=========#=====#============#============#========#============#=============#
```

Component Matrix

```
#============================#==========#
#                            # Component#
```

```
#                            #----+-----#
#                            #  1 |  2  #
#----------------------------#----+-----#
#          Feature inspection# .82|  .35#
#       Cognitive walkthrough# .75| -.31#
#           Expert evaluation# .81|  .17#
#        Think aloud approach# .71| -.61#
#        Heuristic evaluation# .61| -.64#
#Perspective based inspection# .68| -.66#
#          Other - User trial# .60|  .48#
#       Other - User feedback# .76|  .52#
#                Other - None# .79|  .49#
#============================#====#=====#
```

Appendix K: Correlation matrices

Correlation matrix for usability aspects

Table 66: Correlation matrix for usability aspects

		A	B	C	D	E	F	G	H	I	J
Correlation Coefficient	A	1.000									
Sig. (2-tailed)											
Correlation Coefficient	B	.456*	1.000								
Sig. (2-tailed)		.010									
Correlation Coefficient	C	.367*	.340**	1.000							
Sig. (2-tailed)		.010	.000								
Correlation Coefficient	D	.371*	0.403*	.272**	1.000						
Sig. (2-tailed)		.010	.000	.045							
Correlation Coefficient	E	.048	.440*	.245*	.157*	1.000					
Sig. (2-tailed)		.448	.000	.049	.047						
Correlation Coefficient	F	.146*	.303**	.235*	0.310*	-.030	1.000				
Sig. (2-tailed)		.050	.030	.040	.026	.640					
Correlation Coefficient	G	.261*	.060	.119	.105*	-.172**	.075	1.000			
Sig. (2-tailed)		.045	.339	.058	.049	.006	.237				
Correlation Coefficient	H	.475**	.396**	.232*	.106	.128*	-.092	-.044	1.000		
Sig. (2-tailed)		.000	.002	.044	.091	.042	.144	.482			
Correlation Coefficient	I	.027	.426**	.206*	.083	.151*	.041	.212**	.437**	1.000	
Sig. (2-tailed)		.665	.000	.040	.190	.051	.515	.001	.000		
Correlation Coefficient	J	.086	.062	.262*	.045	.128	.136*	-.063	.197**	.037	1.000
Sig. (2-tailed)		.173	.325	.039	.477	.072	.030	.317	.002	.558	

**- Significant at 95% level*

***- Significant at 99% level*

Correlation matrix for usability capturing methods

Table 67: Correlation matrix for usability capturing methods

		A	B	C	D	E	F	G	H	I	J
Correlation Coefficient	AA	-.085	.024	-.089	.249**	.112	.146*	.599**	-.012	.104	.171
Sig. (2-tailed)		.176	.705	.159	.000	.075	.020	.000	.846	.100	.056
Correlation Coefficient	AB	-.105	.001	.064	.017	-.023	.026	.086	.097	.259**	.006
Sig. (2-tailed)		.095	.985	.314	.790	.714	.680	.175	.123	.000	.701
Correlation Coefficient	AC	-.072	.302*	.015	.134*	0.118*	-.113	.225**	-.039	.147*	.038
Sig. (2-tailed)		.251	.038	.000	.033	.050	.074	.000	.540	.019	.542
Correlation Coefficient	AD	.171**	.441**	0.078	.022	.039	.361**	.075	.122	.138*	.047

Sig. (2-tailed)		.040	.000	.065	.723	.538	.000	.236	.052	.028	.460
Correlation Coefficient	AE	.086	.159*	.026	.065	.166**	.031	-.002	.013	.103	.023
Sig. (2-tailed)		.172	.011	.682	.307	.008	.629	.978	.842	.001	.000
Correlation Coefficient	AF	.131*	.117	-.059	.256**	.136*	.184	.094	.440**	.486**	.159*
Sig. (2-tailed)		.037	.064	.351	.000	.030	.063	.137	.000	.000	.011
Correlation Coefficient	AG	-.055	.058	.030	.126*	-.104	.150*	.225**	.052	.193**	.197**
Sig. (2-tailed)		.379	.362	.634	.045	.098	.017	.000	.414	.002	.002
Correlation Coefficient	AH	.046	.013	.050	-.040	.444**	.178**	.276**	.123	.300	-.065
Sig. (2-tailed)		.465	.835	.424	.522	.000	.005	.000	.071	.000	.304
Correlation Coefficient	AI	.258**	.206*	.376*	.130*	-.077	.251**	.015	.190**	.061	.098
Sig. (2-tailed)		.000	.010	.015	.038	.222	.000	.070	.002	.335	.120
Correlation Coefficient	AJ	-.047	.409**	.080	-.002	-.129	.015	-.035	.079	.162	.033
Sig. (2-tailed)		.452	.000	.204	.976	.070	.810	.576	.080	.060	.600
Correlation Coefficient	AK	.037	.494**	.194	.030	.056	-.170**	.013	.084	-.032	.055
Sig. (2-tailed)		.563	.000	.082	.632	.373	.007	.501	.182	.613	.414

Correlation matrix for usability testing methods

Table 68: Correlation matrix for usability testing methods

		A	B	C	D	E	F	G	H	I	J
Correlation Coefficient	AAA	.105	.108	.016	.130	.048	-.405**	-.165**	.225**	-.215**	.125*
Sig. (2-tailed)		.010	.012	.799	.048	.446	.000	.009	.000	.001	.046
Correlation Coefficient	AAB	.035	.014	.250**	.139	.176**	-.060	.101	.055	.259**	.113
Sig. (2-tailed)		.581	.821	.000	.058	.005	.345	.110	.381	.000	.074
Correlation Coefficient	AAC	.010	-.014	-.007	.135	.207**	-.005	-.039	-.088	-.012	.197**
Sig. (2-tailed)		.880	.821	.915	.023	.001	.935	.534	.163	.844	.002
Correlation Coefficient	AAD	.155	.292**	.168	.132	.112	-.271**	-.027	-.054	.157*	-.124*
Sig. (2-tailed)		.214	.000	.107	.046	.075	.000	.672	.394	.012	.049
Correlation Coefficient	AAE	0.204	.103	.093	.088	.069	-.197**	.230**	-.154*	.207**	.140*
Sig. (2-tailed)		.099	.062	.141	.161	.273	.002	.000	.014	.001	.025
Correlation Coefficient	AAF	.195**	.113	.211	.193	.206**	.214**	-.039	-.067	.245**	.275**
Sig. (2-tailed)		.002	.11	.061	.100	.001	.001	.542	.286	.000	.000
Correlation Coefficient	AAG	.129*	-.028	-.109	.080	-.036	-.047	.137*	-.178**	-.148*	.040
Sig. (2-tailed)		.041	.663	.085	.205	.566	.461	.029	.005	.019	.529
Correlation Coefficient	AAH	.231**	.229**	.328**	-.084	-.180**	-.171**	-.042	.404**	.084	-.015
Sig. (2-tailed)		.000	.000	.000	.182	.004	.007	.511	.000	.185	.816

Appendix L: Raw interview data

Summary of interviewee responses (translated as necessary):

RESPONSES TO QUESTION 1: Please describe how you manage to capture user requirements for the software you have developed in the recent past?

Interviewee 1

We collect detailed requirements using many methods as we think suitable for the project. Most often we use interviews, focus groups and survey questionnaires, and sometimes we hold workshops, make observations and also develop prototypes if the project is important. We have also started devising a traceability matrix because sometimes there are legal implications of choosing specific requirements.

Interviewee 2

We do detailed planning and collect all the necessary user requirements before we start on the coding for the software. We do everything very carefully. We don't always have time to involve users deeply, so in those cases we rely on the survey method. Otherwise, we try to give potential student and teachers a chance to trial the beta version before we put out the final one.

Interviewee 3

I am new here so I've only worked on improving existing software in our collection. But our development team has a procedure for finding out what users want during the early stage of software development before we start on the programming. I believe the team has done surveys and sometimes held focus groups to find out what users want.

Interviewee 4

We involve users. We ask them what they want. [How?] We ask them directly. [As in an interview?] It's not formal. We invite them. Then we give a brief introduction to the project and they give us their opinions and we take them into account. [Like a focus group?] Yes, but informally.

Interviewee 5

We have a detailed and thorough requirements gathering procedure. [What methods do you

use for this?] We do brainstorming, meetings, we consult all developers for ideas, and sometimes we arrange workshops for large projects. [And these help you to capture *user* requirements?] Yes. [Do you not involve *users* in the process?] Users test the software afterwards and then we make continuous improvements and bring them out in new versions.

Interviewee 6

We ascertain user requirements through careful planning. [Do you involve users in the process?] Our developers work together to determine all the software requirements. Yes, we get instructions what kind of software is required by the management of the university and then we do the research and start the development. [But you don't *involve* users in the process?] You mean students? [Yes, and also teachers, whoever will be using the software.] Not directly, no.

Interviewee 7

We use the PMP method for software development. When we devise the plan, we identify all the requirements and we work to satisfy those requirements as best we can.

Interviewee 8

User requirements, yes, we take care of that at the beginning while we plan for the project. [Does that process involve the users?] Yes, sometimes we ask the users for their suggestions, it depends on the project and how the managements wants us to proceed with it.

Interviewee 9

User requirements... for user requirements, we ask the users directly. Sometimes this is important for large projects, then we ask users for what they would like to see in the software, and we try our best to accommodate their wishes as far as possible. [How do you find out what they want?] We use surveys mostly. Focus groups and interviews are used sometimes too, and on occasions, we've held workshops for large scale projects.

Interviewee 10

All the necessary requirements are gathered upfront as this helps us ensure we meet all those requirements during the programming. [Does the process involve users?] User requirements, of course, because we develop the software for users so they are all user

requirements. [Do you ask the users what they want?] Yes, sometimes we do that because we have to make sure the software meets their needs. [But not all the time?] Um, depends on the software I think.

Interviewee 11

Our software development team follows strict procedures from defining the software parameters and gathering requirements through to completion. When the requirements are gathered, user requirements are captured through brainstorming what we might need and asking potential users directly what they think they need. [How do you ask users?] We ask them, in an informal manner, and they tell us, and we take note of it, but what we incorporate into the design, is decided by the senior members of the team.

Interviewee 12

We capture user requirements very carefully before we start to develop the software. [And, how do you do that?] We carry out research to document everything we might need to include in the design. [Do you ask students for what they might need and if so, how?] Yes, the users tell us usually by providing suggestions and giving their responses to a questionnaire. Occasionally, we choose to ask them directly if it's an important project. [As in interviews?] Yes.

Interviewee 13

We invite the users to give us their thoughts and we consider these suggestions carefully in our design. [How do you invite them?] We use a variety of methods. For example, we did some surveys in the past, and sometimes we ask them together in groups.

Interviewee 14

User requirements, we capture them by talking to our users to know what they need. [How do you talk with them.] We invite anyone interested in using the software and ask for their suggestions at a time when we can take their suggestions on board.

Interviewee 15

We use PMP to develop our software. We capture user requirements by undertaking research before we start to make the program.

Interviewee 16

We find out the user requirements by doing analysis of what future users might need. {How?] We find out from a sample of users and do a survey.

Interviewee 17

We are given the requirements and we have to work with those requirements to carry out the project according to them. [Who makes the list and how?] The senior members of the team get together and decide what requirements to have.

Interviewee 18

Our research team finds out what the users want to have in their software, then we make the software. [How do they find this out?] Sometimes they get instructions from the clients and sometimes they invite the users to discuss what they want.

Interviewee 19

Depending on the software to be developed, we carry out a survey and sometimes interviews in groups.

Interviewee 20

First we do the plan. We find out the requirements by asking the users directly and sometimes by means of a survey and we develop the software accordingly.

RESPONSES TO QUESTION 2: What do you understand by the term usability?

Interviewee 1

I understand usability to mean the ability to use a software as one would like it. For example, it should be easy to use and not complicated. But it should also be enjoyable, and reliability is very important too because it must work as it is supposed to.

Interviewee 2

Usability is about how well the software can be used by its end users so it is important because the users after all will be using the software in the end. We should strive to make these end users satisfied by making sure the software satisfies their needs. If it does, then it is usable, otherwise not.

Interviewee 3

I think there is a big interest in ensuring software usability these days. Some older programmers don't take it too seriously, but I've learnt to make it a high priority. It is all about making the software usable, for example, in terms of reliability, enjoyability, ease of use, and so on. It is about ensuring the software is designed and operates the way it is supposed to and users can use it well without any problems.

Interviewee 4

It means how good the software can be used. If it can be used properly, it's usable, that's it.

Interviewee 5

I understand it to mean the software can be used by its users in a way that is easy, efficient, and enjoyable at the same time.

Interviewee 6

Usability refers to how well the software can be used, and this has many aspects. For example, if it is easy to use, and it can do what is designed to do, then the software usability is good.

Interviewee 7

Usability... it's about making the software usable. [And, what makes a software *usable*?] If it can be used, if it works and it works well, it is usable.

Interviewee 8

It's about how useful a software is. [Useful for whom?] For the users of course.

Interviewee 9

Usability is about things like reliability, efficiency, how easy it is to use, and how much it is liked. So usability encompasses a number of things. In short, we can say it is about how usable the software is by its end users.

Interviewee 10

It concerns certain aspects of the software design that makes the software highly usable for its users. [What kind of aspects?] Like, it should work first of all. It should be easy to use. It should basically do what the users want it to do.

Interviewee 11

Usability is about the capability of the software to do what it is supposed to do. [What might that include?] If it meets the requirements, if it's engineered the way it was planned, if it works and it works efficiently, I suppose that makes the software usable and that's what usability is about.

Interviewee 12

Usability, it's a term that's used to describe how well a software can be used. It's a broad term though. It encompasses such aspects as reliability, efficiency, how easy it is to use and remember, compatibility, how pleasurable it is to use, and other such aspects.

Interviewee 13

Usability is all about making software fit for use. It must work properly of course, but it should also be enjoyable and easy to use.

Interviewee 14

It is about how good is the software to be used. [In what ways?] In how it looks, how it performs, how well it is made and works.

Interviewee 15

How well a software can be used is usability, I'd say. [In what way?] If it works well, if it can do what it's supposed to do.

Interviewee 16

Usability defines the extent to which the software can be used easily and reliably.

Interviewee 17

The software must be usable of course. How much it is usable is what usability is about.

Interviewee 18

If the software is made well and the users can comfortably use it they way they expect, usability requirements are met.

Interviewee 19

Usability dictates aspects such as how reliable and satisfactory the software is, and how easy it can be used for what users want to do with it.

Interviewee 20

I understand usability to be about things like ease of use, extent of satisfaction, adequate performance of the software, and so on.

RESPONSES TO QUESTION 3: How well do you think your software has been able to ensure usability in terms of how well users are able to learn to use the software?

Interviewee 1

We try our best to make the software easy to learn to use by making it organised logically and asking some potential users to try it. If the project is important and we develop a prototype, this gives us the chance for more users to try out the software before we create the real version and we ask them for feedback.

Interviewee 2

I think we are in a very strong position of always ensuring usability due to our detailed planning. The trials make it possible to test how well the software can be learnt quickly and to make any changes to improve things if they need improvement.

Interviewee 3

Our team focuses on usability and one area we work hard on is making it easy to learn to use. Normally, we use familiar elements for the interface and procedures from the operating system and common programs as this makes it easy for users. The few non-familiar aspects are carefully organised to make it intuitive and minimise the possibility of erring.

Interviewee 4

When we ask the users, I suppose that goes some way to making the software usable. Learning to use, they can only do that after it is developed. [So how do you ensure they can learn it easily whilst at the developing stage?] The way we design it, we do our best it can be used easily.

Interviewee 5

When we plan for the software, we take care of such factors to make sure the end users can use the created software easily as far as is possible. The continuous improvements also allow us to make refinements so eventually it becomes as easy to use as possible.

Interviewee 6

We plan very carefully as I said, and if the software is made to be easy to use then the students can learn to use it easily.

Interviewee 7

If the software is made with familiar elements and in a consistent manner, then the end users can learn to use it easily.

Interviewee 8

I think we make terrific software and it proves to be useful, yes. If it's well-designed, then it is more likely to be useful.

Interviewee 9

I'm confident we produce software that is highly usable in a way that end users can learn to use it easily. We design all elements carefully with the consultation of the end users and this ensures they can learn to use it.

Interviewee 10

When we make the software, all the requirements are ascertained and design considerations are made upfront. Users can learn to use it, this is ensured from the start. We also solicit feedback so users can tell us if they face any issues. Then we keep improving the software all the time.

Interviewee 11

Due to the intensive planning and detailed design, I am confident the end result is software that users can learn to use easily, yes.

Interviewee 12

Software should be designed so that it is easy to learn to use. We are very careful in how we design our software and we take usability issues seriously. During the development, we research all we need to know to produce software that is highly

usable. This includes making sure it is easy to use.

Interviewee 13

They can use it easily if it is made efficiently. Our team makes the software easy to learn to use.

Interviewee 14

It can ensure it is easy to use because we make the software after careful consideration and planning.

Interviewee 15

PMP is an excellent method for software development. Whenever we make software, our users can use it easily.

Interviewee 16

Yes, I believe our software is easy to learn to use. Because we ask the users what they want and they tell us, it is made in a way that suits them best.

Interviewee 17

I am confident that users can learn to use it. Our research team does a good job of finding out what users want.

Interviewee 18

When we are told what requirements to satisfy, and we produce the software accordingly, it is made easy for users to use because they told us how they want it.

Interviewee 19

We ensure usability of all our software as we make it in such a way for users they can use it easily.

Interviewee 20

Usability aspects are ensured because when we involve the users in the design, we make it according to their requirements and we use familiar elements in teh design.

RESPONSES TO QUESTION 4: How well do you think your software has been able to ensure usability in terms of how well users are able to use the software effectively?

Interviewee 1

I think because we involve users to make sure they can easily use the software then it is assured it can be used effectively. Their trial and feedback helps us to make improvements so that by the time the software is completed, it can be used as effectively as possible.

Interviewee 2

Our planning process maps out all possible ways the software can be used and we then ensure that all these ways can performed effectively. The involvement of users during trials helps to check the effectiveness of each task before users have to do them for real.

Interviewee 3

The involvement of users in the development process is the best way to ensure this otherwise if they are not involved we can never be sure the software is designed well for users to use despite our best efforts. We conduct thorough testing of the software and this ensures usability.

Interviewee 4

As I said, when we design the software, we think about all these things, and we do our best to ensure when it is finished, the software can be used effectively.

Interviewee 5

We make sure of all aspects of usability during the design stage, and if there is any lacking there is always scope for further improvement in the revised versions.

Interviewee 6

Effectiveness of the software, when we do the plan, we take care of all these aspects of usability. The software is make so that it is easy and effective in use.

Interviewee 7

The method we use is very detailed so the software is bound to be effective otherwise it would not be made available. We always ensure effectiveness this way.

Interviewee 8

It has to be effective. If it's well-designed, then it would have to be effective, so we always ensure we make effective software because we plan it properly before making it.

Interviewee 9

Effectiveness is quite well insured as well. When we plan the design, all important details are taken care of and the development process makes sure the software is made to be effective, efficient and reliable.

Interviewee 10

Using it effectively is similarly made sure of through ascertaining all the necessary requirements and programming to make sure those requirements are met.

Interviewee 11

Effectiveness is ensured because I said we do the plan and gather the requirements and make the software accordingly.

Interviewee 12

Yes, effectiveness is important. Again, we ensure this and all other important aspects of usability by how we design our software.

Interviewee 13

It is effective because we make the software ourselves. We make it very carefully.

Interviewee 14

Effective use is assured because we plan it carefully and make sure it will be effective as possible.

Interviewee 15

PMP is a very effective method for software development. When we make the software using PMP, it is very effective.

Interviewee 16

By carefully taking our users' needs into account, the software is made effective because it is designed precisely according to those requirements.

Interviewee 17

The research team does a wonderful job of knowing how to make the software effective. And our development team satisfies those specifications in every detail.

Interviewee 18

Our software is effective. The instructions to make them are always clear and we don't finish the job until the software is developed to our satisfaction.

Interviewee 19

Effectiveness of our software is guaranteed. We develop the software very carefully to ensure usability.

Interviewee 20

If users can use the software effectively then the software is usable. Their feedback and our continuous development cycle helps to improve the software.

RESPONSES TO QUESTION 5: How well do you think your software has been able to ensure usability in terms of how well users are able to to use the software reliably?

Interviewee 1

We give high priority to reliability of all our software. As I said, the trials and feedbacks are useful for this purpose because if there is any problem, it can be spotted early and we can improve the software. We work in teams to make sure the software works well as it is supposed to be, and the user input helps so that the final version runs smoothly.

Interviewee 2

Planning ensures reliability is maintained throughout the software development life cycle. Reliability is important so that the software can be used for which it is designed. The trial stage makes sure users are satisfied with this aspect of the software.

Interviewee 3

Reliability is a very important consideration in software development. By involving users and otherwise, the software is checked thoroughly before the final version is released for guaranteeing its reliability.

Interviewee 4

We make sure of all these factors when we design the software. We don't release a software until we are satisfied it is reliable because if it is not reliable, I don't think we can justify releasing it.

Interviewee 5

In the same way, the planning process we conduct with great care and detail and the further improvements we continue to make are targeted at making the software usable, easy, effective and reliable, and so on.

Interviewee 6

Of course, reliability is very important. Everything is considered during the planning. Also, we always make more improvements so reliability is always improving and sometimes we have to update the software in line with the latest technologies.

Interviewee 7

Our planning is very detailed, so the software is always going to be reliable, otherwise as I said, it would not be given to the end users. If there is any lacking in reliability then this is important to fix.

Interviewee 8

The same I would say. When we make the software, it is all planned carefully to make sure it is effective and reliable.

Interviewee 9

Reliability is also insured in this way. The development process makes sure what we produce is efficient and reliable as far as is practically possible. The development does not end until out team is sure we have produced software that is capable of being used reliably.

Interviewee 10

I don't think reliability is ever a problem because the entire development process ensures something as important as reliability is upheld. The software has to be reliable otherwise it would not be released for use until it is.

Interviewee 11

The software is definitely reliable all the time. We make sure of that from start to end as we plan and develop the software with the help of experienced developers.

Interviewee 12

Reliability is no doubt a very important factor for ensuring usability. When we create the software, we strive to make it as highly reliable as possible until we are satisfied it works the way it is designed to work.

Interviewee 13

The software we make is reliable. We make sure of that by the way we make it very carefully.

Interviewee 14

We ensure our software reliability is good. We always plan and finish when it is ready to use.

Interviewee 15

As you know, in PMP we plan for the software very carefully. Reliability is an important feature of a software and the careful planning makes sure our software is always highly reliable.

Interviewee 16

I believe our software is as reliable as we can possibly make it because a lot of time and attention goes into making it. Then there are always improvements that are ongoing.

Interviewee 17

Our research team does a good job of ensuring usability and helping us to make highly reliable software. I'm confident of that.

Interviewee 18

If you see the planning that goes into developing our software, I'd say reliability is a key priority because obviously the software has to work.

Interviewee 19

We make very reliable software because we only release it when we are confident it has met the requirements and works perfectly.

Interviewee 20

Reliability is a very important factor. Our software is very reliable. We have a good team for development and testing and if anything gets left out, we can always fix things immediately.

RESPONSES TO QUESTION 6: How well do you think your software has been able to ensure usability in terms of how well users are able to to use the software in an engaging manner?

Interviewee 1

Yes, we do try our best to make our software enjoyable, but it depends on what software it is, and our main priority is making sure of reliability and efficiency. When we ask users for feedback, we also ask them how much they liked using the software, so yes it is also important.

Interviewee 2

The software planning process is mostly concerned with things like reliability, effectiveness, usability. But I think it is sometimes necessary to also make sure the software can be engaging for users, especially if they are children or it is an entertainment software. For e-learning software too, it can help to motivate the users to learn. As I said, we provide opportunities for trialling the software, so users can report if they don't like anything.

Interviewee 3

User involvement also ensures usability in terms of the ability to engage as this leads to overall user satisfaction. So if users indicate their satisfaction, the software can be assumed to be successful in engaging its users.

Interviewee 4

Engaging, well, I think when we ask the students during what you could call a focus group, the students tend to focus more on elements that make the software appealing to them. So I think that ensures the software being engaging very well.

Interviewee 5

Yes, engagement is important too. Our focus when creating e-learning software is on ensuring that learning is effective, but we do try to achieve this in a way the students can enjoy depending on what kind of software it is, the age of the student, and so on.

Interviewee 6

Engaging for users... I think that's more important for e-learning software for younger school age students. We make software for university students. [So is it not necessary to make the software engaging for them?] Learning is more important for them, so I think there are higher priorities.

Interviewee 7

Software can be engaging also. You pointed out effectiveness and reliability. I would say these are most important first of all to make the software usable. And if possible then it can engaging also.

Interviewee 8

Engaging, well. Engaging for students? [Yes, but also all other users as well.] We make it engaging too I suppose. I find it engaging, yes.

Interviewee 9

Our consultation process with the end-users makes sure of that too. For us, things like reliability and efficiency are most important, but the input from end-users is often useful in identifying other such aspects as making it more attractive and enjoyable.

Interviewee 10

Making the software engaging is a concern we also take into account but not as important I would say as reliability. When users tell us what they want, it ensures the software is made engaging as well.

Interviewee 11

The software we produce is engaging. We make sure of that when we ask the users for what they would like to see in the software and how it should look and work.

Interviewee 12

You could say the same thing for trying to make it engaging. Our design elements are thought out carefully so that the software can be used not only reliably but also to provide a pleasurable experience. In the context of e-learning, I think this is important because it makes the learning easier if the software is enjoyable.

Interviewee 13

Engaging, yes. We make engaging software for our users.

Interviewee 14

When we design the software, we make it attractive and the users like it how we make it.

Interviewee 15

Engaging is not a high priority I think, but maybe for young children's software.

Interviewee 16

In the survey of users, the users point out what they want. There's a lot of focus on making software attractive these days and we do our best to accommodate their wishes.

Interviewee 17

We get our instructions from the research team. They tell us what to make and how to make it and we make it as instructed. It is as engaging as we can make it.

Interviewee 18

I think it's engaging for users. We focus more though on things like reliability and for attractiveness we do our best.

Interviewee 19

Our software is made to be attractive for users because many of our users are students and we have to make them motivated.

Interviewee 20

Making engaging software is important as well. Our software is always engaging and reliable.

RESPONSES TO QUESTION 7: How do you test for usability afterwards?

Interviewee 1

You mean after the software is finished? [prompted: yes] We do more testing depending on the importance of the software and how much time we have for this. The testing we do is by involving users and asking them to try the software so we see how good it is made. We always check for reliability and efficiency especially. We only finish the software when we are satisfied with these two qualities especially but most of the time, we have very short time and we have to do things quickly. [prompted: yes, but what about afterwards, after the software is completed, do you test for usability again?] No, we only complete it if it is ready and the users are happy, then it is finished and we have to work on other projects also.

Interviewee 2

We are concerned about usability from the very beginning. During the development, the trialling then ensures usability, and we can be satisfied that it is indeed so. Afterwards, we sometimes do further testing while continuing to work on ongoing projects to make further improvements in new versions. [promoted: But *how* do you test for usability later?] When a project is ongoing, we use the same methods at intervals, like I mentioned before such as surveys.

Interviewee 3

I think we do have a procedure for post-release usability testing as well but I'm not sure about it. Even then, I think it is probably only done for some software, maybe not all. You see, we test for usability before the final version is released.

Interviewee 4

You mean after it is developed? [Yes] We don't release software until we are satisfied it is complete and ready to be used. When we involve the students, it is all for ensuring

usability and their satisfaction.

Interviewee 5

Well, we make sure of usability throughout the entire development process. By the time development is completed, this ensures the software is highly usable. [But do you test again after the development is completed?] We are always testing and making improvements. If it is a big and important project, we continuously improve the software and each phase of testing helps make these improvements.

Interviewee 6

Our testing is done from start to finish. When we develop the software, we are always making sure the software is made well. [What about *after* it's finished?] Yes, when it is completed, we test the software again and only when we are satisfied it is ready then we make the software available to the users.

Interviewee 7

Our software is developed after detailed planning. This makes it ensure usability from the beginning. Over the course of development, the usability is always improving and by the time it is finished being developed, the software is made available for end users only when we are satisfied with its performance.

Interviewee 8

After what, you mean after the software is finished? [Yes.] Um, when it's done, it's done. There's no need for any more testing. We have lots of projects in line, so we have to move on.

Interviewee 9

Afterwards, as in after its release? [Yes.] Usability testing is mostly conducted towards the end of the development life-cycle before the release so we can make any necessary final modifications. Certain projects, which are ongoing, for them we continuously make improvements as we release new versions, as is the case with our university's main e-learning software, which is in-house developed and maintained.

Interviewee 10

After the completion of the software you mean? [Yes.] A software project is only

completed when we are totally satisfied it is fit for productive use.

Interviewee 11

We always test the capabilities of the software we make from start to finish. [Do you test again after it is finished?] It is not finished until it is tested and approved for use. It is only ready when we are sure it is highly usable.

Interviewee 12

Usability testing is conducted both during and after the development and release of the software. Many of our projects are maintained and updated so usability testing is conducted on occasions or when there is a need to conduct it. [How do you test for usability?] We create a set of performance tasks unique to each software and we see how well each of those tasks is performed before and after any changes in the code. We also use Morae software to test for usability for some projects.

Interviewee 13

What you mean by after? [After the project is finished.] But we only finish when it is tested and everything is okay.

Interviewee 14

We work very hard and test it all the time and when it is approved as usable we give the software to our customers.

Interviewee 15

PMP is a plan based method. We plan first and then make the software according to the plan. Our software is always usable.

Interviewee 16

The survey helps us to know what users want so the software is usable when we make it. [Do you test for this afterwards?] If necessary, we ask the users if they are satisfied, yes.

Interviewee 17

The research data ensures a high level of software usability. The researchers make a careful analysis of what is needed to get it right first time.

Interviewee 18

Usability testing is done during the development of the software. We work together to test it and produce it how it is needed.

Interviewee 19

After the completion? [Yes.] Well, it is ongoing development so when we get the feedback that improvement is required or we test and find a bug ourselves, we work on it.

Interviewee 20

We release the software after all the testing is done. When we are satisfied, we release it. Of course, sometimes a problem is only noticed after a long time of use. In this case, we make the improvements in the next cycle depending on what kind of software it is.

RESPONSES TO QUESTION 8: Are you satisfied with your existing procedures to capture software requirements for usability design?

Interviewee 1

Yes, we are very satisfied because we spend a lot of time and we gather the important requirements very carefully and we have priority for making our customers happy.

Interviewee 2

Yes, definitely. We engage in careful planning and we also involve users to make sure of usability.

Interviewee 3

I believe our team has developed a fairly satisfactory procedure for developing in-house software and usability concerns are also addressed because users are asked directly for what they need and we only proceed after all the necessary requirements have been captured.

Interviewee 4

Yes, I think so. We make sure our software is well developed before releasing it.

Interviewee 5

Yes, of course. Our existing procedures are well-grounded and well-planned. Our development team is strong and we take care of every little detail to make the software of a high standard and usable.

Interviewee 6

Yes, I am satisfied because we have a very good team of developers and we take care of the whole development process to produce high quality software.

Interviewee 7

Definitely. We've been making software for many years and our in-house procedures capture requirements very carefully and we always ensure usability and improve the usability until the software is ready to be given to the end-users.

Interviewee 8

Unequivocally, I am, yes. We make terrific software that is always efficient, reliable, and useful because we design it and make it carefully.

Interviewee 9

We capture most of the requirements including all the essential ones at the initial stages but some get added during later phases. Our consultation with end-users, careful planning, adoption of quality standards, management involvement, we do our best we can to produce software that satisfies end-users' requirements.

Interviewee 10

Yes. I would say so.

Interviewee 11

Oh yes, definitely. I've been working for a long time in this place. Our procedures are very well detailed and all development projects are documented.

Interviewee 12

Yes. I am satisfied with our existing procedures for gathering requirements because they work well for us.

Interviewee 13

Yes.

Interviewee 14

Yes, I am.

Interviewee 15

Yes, certainly.

Interviewee 16

Yes, we are satisfied. The users are involved and we make it how they want and they get it.

Interviewee 17

Yes. The researchers put a lot of effort to make sure all the requirements are met.

Interviewee 18

Yes.

Interviewee 19

I believe our existing procedures are working well.

Interviewee 20

Yes, I am. The software requirements are ascertained and the software is made accordingly until the software is developed to our satisfication, so yes.

RESPONSES TO QUESTION 9: Do you think there is scope for improving the way software requirements are captured?

Interviewee 1

I think there can always be scope for being more careful but it really depends on how much time you have so we make priority, but our method, yes I am satisfied.

Interviewee 2

I am satisfied with our software development team's efforts. We work very hard and spend a lot of time in the planning to make sure everything is as perfect as possible. [prompted: but do you think we can improve things further?] Maybe.

Interviewee 3

As I said, I'm new on the team here, and I think we're doing well, but I'm receptive to finding ways to improve the way requirements are captured, yes.

Interviewee 4

There is always scope for improving software I would say. It depends on what you want to make and how much time and investment you are willing to put into it.

Interviewee 5

Our planning is done very carefully and thoughtfully, and we make sure the software is made to very high standards. [But, can this be improved further, especially in terms of how the *requirements* are captured?] When we do the planning, we make a list of all the software requirements; the things we need in the software and for the software to be able to do. Then we make continuous improvements to the software.

Interviewee 6

I am satisfied with the performance of our development team. We work very hard on all our software projects. We consider all the necessary requirements from the start and then we develop according to those requirements very carefully.

Interviewee 7

We capture all the software requirements at the beginning. We do this very carefully. I am satisfied our in-house development team does this very well.

Interviewee 8

I agree, you can always improve something of course, if you give it sufficient time and attention. But I think we devote enough time and attention to make it as perfect as possible first time round.

Interviewee 9

Scope for improvement, I'm satisfied with our development efforts. I think our team spends a lot of time and effort already. I'm confident our requirements capturing process is adequate to produce the software that end-users want.

Interviewee 10

We're doing well. Maybe we can improve some things here and there, but overall we're okay.

Interviewee 11

Improvement is possible, I don't rule out that possibility, but I'm satisfied we're doing a good job of ensuring we produce software that meets all user requirements.

Interviewee 12

Scope for improvement cannot be ruled out. You see, a lot depends on what type of software you are developing as some will require more planning and detail of design than others. Our requirements gathering procedures are thorough but they are also tailored to the specific requirements of the project at hand.

Interviewee 13

Software requirements, when we ask the users they tell us what they want and we make the software according to that exactly.

Interviewee 14

We work hard to develop software and it is good for use when it is finished. [But could things be improved more?] I think we are doing very good because I know how hard we work on our software.

Interviewee 15

I think PMP does a good job already. I am satisfied.

Interviewee 16

The survey we do helps to find out exactly what the potential users are most likely to want. I think the survey is adequate to find out the requirements.

Interviewee 17

I'm confident in the ability of our research team because they work very hard to make us develop the software as per the requirements.

Interviewee 18

I'm happy with the efforts of all members of our development team. We make high quality

software according to the requirements.

Interviewee 19

We capture requirements at the start and we ask the users what they want so the software is well developed.

Interviewee 20

There can always be improvements. It depends on what you want to do and how much you are willing to invest in terms of time and dedication. But I am confident we capture requirements very well already.

RESPONSES TO QUESTION 10: Would you be willing to improve this requirements capturing process if it is possible to do so?

Interviewee 1

Yes, we are always willing to learn new methods and making improvements because we should always try to become more efficient and make customers happy and technology changes also mean sometimes we have to change along with it. But what do you have in mind? [prompted to next question]

Interviewee 2

The management dictates what tools we use and what procedures we adopt and we apply and follow these very carefully. If you're suggesting there are ways to improve our processes, then we would need to discuss that with the management.

Interviewee 3

Certainly, but it would have to be approved by the whole team. How do you suppose we could improve the requirements capturing process then? [prompted to next question]

Interviewee 4

If possible, why not?

Interviewee 5

Our development team is very experienced and we have developed an expertise over the

years in planning and development. We continuously improve the software and sometimes we refine our procedures as well according to the task at hand.

Interviewee 6

I am satisfied with our efforts. It is already a very time consuming process. I don't think there is need for adding more elements to it. [What about *improving* it, maybe streamlining it, not adding more to ti?] Maybe, but I think we are all satisfied with the existing process and as a busy team, I'm not sure if the management would be willing to risk changing or even trying some other method.

Interviewee 7

When we capture the requirements, it is carefully thought out and we focus on what's important for the software to be able to do. Our management has defined the process very well for us. Personally, I am open to ways for improvement, but for in-house development, any changes to the existing procedures would have to be approved.

Interviewee 8

How are you supposing to improve this requirements capturing process? [I'm only asking if you would be willing, if it would be possible?] It's not my decision. We have to follow orders. You will have to discuss that with our superiors.

Interviewee 9

We do review our procedures once in a while or when it become necessary for a certain project, so we adapt as well. Our procedures are tried and tested, but we do keep adjusting them in line with the needs.

Interviewee 10

We can always improve things, but the way they are at present, we're doing pretty well.

Interviewee 11

It's not down to me to make any changes as I said we follow strict procedures, but the management occasionally advises us specifically on how to handle particular projects.

Interviewee 12

If possible? You're suggesting you have a magic solution for gathering requirements? [No,

just some ideas that you might find useful, a special methodology in fact that is focused on usability.] I see. There are many methods in software design, I know. Which one we use though comes down to the project at hand and the conditions under which the software is to be developed. [So would you be willing to try another methodology?] I'm certainly interested to know what you have in mind. Go ahead.

Interviewee 13

We make the software in the way the users want it and it works good for them.

Interviewee 14

I am happy we make very good software. Our team works very hard for that.

Interviewee 15

PMP is a great method for creating software and we have highly professional PMP qualified developers in our team. I think we manage to capture requirements well.

Interviewee 16

When we ask the users themselves, we capture as much as they want as possible and the work is done to meet all the important requirements.

Interviewee 17

I think we're doing well as it is. It can become very busy sometimes. I don't think we need to try another method. We capture requirements fine.

Interviewee 18

The researchers are doing a good job of finding out the requirements and now and then they improve their method. We are responsible for making the software only.

Interviewee 19

I think the way we capture requirements is going well. The software is acceptable to us and also acceptable to our users.

Interviewee 20

The decision to change the method rests with the management. It is always possible to improve things, but I am satisfied we are doing a good job of it.

RESPONSES TO QUESTION 11: Are you aware of Usage-Centred Design and its possibilities for improving software usability?

Interviewee 1

Usage Centred Design, no, but we have studied lots of methods and we apply PMP methods for developing our software and follow our management instructions. We have to make the software as the management tells us and the priority is to make sure we make reliable and efficient software.

Interviewee 2

No, I've never heard of that, but I know there are lots of different methods and approaches in the field. Some newer methods do not advocate much planning or none at all, but I think it is important to ensure the development process runs smoothly. I know many newer methods focus a lot on usability, but I'm satisfied we involve users enough to make sure the software we develop is highly usable.

Interviewee 3

I've heard of User-Centred Design, yes, but I'm not sure about Usage if that's what you said. [prompted: yes, *Usage*-Centred Design]. Oh, well I know there are many methods in Software Engineering, about plan-based and agile methods, user-centric methods and User-Centred Design. Maybe Usage-Centred Design is a type of User-Centred Design? [Briefly introduced UgCD to the interviewee because he was interested to know about it.] Oh, I see, it's like, so you have some planning as well and also make it agile? [Yes]. Well, it seems interesting and if it can improve usability then it's worth looking into I suppose.

Interviewee 4

You mean User-Centred Design? [No, Usage-Centred Design] No.

Interviewee 5

No, I haven't. [What about User-Centred Design] Yes, but we use our own method which is based on PMP.

Interviewee 6

Usage-centred, no. I know there are many theoretical methods, but at the end of the day, a software house has to work in a way that suits them and their work best, and I think our team is doing well already.

Interviewee 7

We use PMP. I have not heard of Usage-Centred Design before.

Interviewee 8

No, I'm afraid, I don't know anything about that. It's the first time I heard it.

Interviewee 9

Usage-Centred Design, no. No, I haven't, I'll admit that. For improving usability, I'm sure there are so many different methods out there. In the end, we do what suits our needs the most though. What's special about Usage-Centred Design then? [Briefly introduced UgCD and its potential for improving software usability.]

Interviewee 10

No, I never heard of it before I think.

Interviewee 11

Usage, no. Sorry.

Interviewee 12

No. You said Usage-Centred, am I right, not User-Centred? {Yes, Usage-Centred.] I haven't, no. Is this the usability based method you were referring to? [Yes.] I haven't heard anything about it before, but I would guess defining and testing tasks would feature in this method. [Yes, they do.] Ah, see I have some idea of what it might be like already. By all means, you can tell me more about. We focus a lot on usability, so we could always learn more from new ideas. {Gave some more in-depth details of UgCD due to the interest.]

Interviewee 13

No, I don't know.

Interviewee 14

No, we only use the method we are told to use.

Interviewee 15

Everyone uses PMP. It's a well known method. There may be other methods, but PMP works great for u.

Interviewee 16

No, I never heard of it before.

Interviewee 17

Maybe our research team knows about it, but we only make the software how they tell us to make it.

Interviewee 18

No. Some other members of the team might know, but not me.

Interviewee 19

Do you mean User-Centred Design? [No. Usage-Centred Design.] I don' know.

Interviewee 20

I have not heard of Usage-Centred Design before.

Appendix M: SUS questionnaire

Below is a copy of the SUS questionnaire that was being used by the software developers prior to the UgCD implementation. It consists of 10 questions. This scale is available at http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Appendix N: The post-UgCD survey and interview questionnaire

This post-UgCD survey questionnaire was administered to participants of the meeting at which UgCD was demonstrated *after* the software changes had been made based on the outcome of the meeting.

Post-UgCD Survey Questionnaire

Please indicate your extent of agreement with the following statements about your experience of the trial during the meeting, and your views of the effectiveness of UgCD with respect to software usability.

Table 69: Post-UgCD trial survey questionnaire

No.	Aspect	Strongly disagree	Agree	Unsure	Agree	Strongly agree
1	UgCD helped to focus on usability.					
2	The roles of the users were clearly identified.					
3	The tasks for each user role were clearly identified.					
4	The learnability of the software has been improved.					
5	The rememberability of the software has been improved.					
6	The reliability of the software has been improved.					
7	The efficiency of the software has been improved.					
8	Users are now more satisfied with the software.					
9	UgCD is a useful methodology.					
10	The UgCD methodology was easy to apply.					

Raw data for the post-UgCD survey

Breakdown by category of respondent (developers and teachers)

Table 70: Raw data for the post-UgCD survey

		Developers					Teachers				
No.	**Aspect**	**SD**	**D**	**U**	**A**	**SA**	**SD**	**D**	**U**	**A**	**SA**
	Value >	1	2	3	4	5	1	2	3	4	5
1	UgCD helped to focus on usability.	0	0	0	0	5	0	0	0	0	9
2	The roles of the users were clearly identified.	0	0	0	0	5	0	0	0	0	9
3	The tasks for each user role were clearly identified.	0	0	0	0	5	0	0	0	3	6
4	The learnability of the software has been improved.	0	0	0	0	5	0	0	0	4	5
5	The rememberability of the software has been improved.	0	0	0	1	4	0	0	1	5	3
6	The reliability of the software has been improved.	0	0	0	2	3	0	0	2	3	4
7	The efficiency of the software has been improved.	0	0	0	0	5	0	1	0	3	5
8	Users are now more satisfied with the software.	0	0	0	0	5	0	0	0	2	7
9	UgCD is a useful methodology.	0	0	0	1	4	0	0	1	1	7
10	The UgCD methodology was easy to apply.	0	0	0	1	4	0	0	2	2	5

Appendix O: Post-UgCD interview questions and their replies

The following are the questions that were put to the leading developer during the Post-UgCD interview, which was conducted online using Skype:

1. How well do you think UgCD helped in focusing on usability?
2. How well do you think UgCD helped to improve usability in terms of the five usability aspects we focused on?
3. What do you think are the main strengths and weaknesses of the UgCD methodology for capturing usability requirements?
4. How well do you think UgCD helped to capture requirements more completely and precisely than before?
5. Are you likely to adopt UgCD for your future software development projects?

The following are the replies given by the leading developer (the wording of the main points made):

1. UgCD helped very much in focusing on usability. I think we all agree in the meeting and in my team that it is very usability focused method and it helped us to improve usability very well.
2. Yes, we are satisfied UgCD helps to think of ways for improving efficiency especially and also reliability. I think we can do more to improve its efficiency. We will work more in this area so we can make the software more efficient. [What about the other aspects?] Yes, it is good also for the other aspects. Before, we didn't think really of things like learnability aspect, and yes user satisfaction is important also for users.
3. I thinks its biggest strength is the usability focus, yes. It helped us to make real improvements to the software. It made us very clear on what we need to do and not need to guess about it because when we think of the tasks, then we can think to find ways to make the tasks more efficient. [What about weaknesses?] Maybe, we need to prepare for meeting with users, but when we started in applying UgCD, then together we come up with lots of ideas and lots of requirements.

4. Yes, definitely. It is very good in this way. When we applied UgCD, together we had a good list of usability requirements, and we have something we can find useful for working on the software. [Were these requirements more complete and precise than you expected?] Yes. It was a comprehensive list, and because of the focus on usability, together we could be precise about what we want and what we need for the software to do.

5. It was a good experience for us. I think, it made us think very deeply about usability and efficiency, In future, yes, we can use UgCD process for being clear on the tasks especially, and then thinking of usability aspects. [What about involving and consulting with users, including students?] I think the teachers made very useful contribution in the meeting, and we are using SUS already for students, but yes, I think we can take good advantage of this, and we will involve all users including students so we can make more efficient software.

Appendix P: UgCD artefacts used during the trial

UgCD design notation used for activity modelling during the trial

Figure 45: UgCD design notation used for activity modelling

Symbol	Name	Description
	actor, user actor	activity participant interacting with the system of reference
	role, user role	relationship between an actor and the system of reference
	system actor	non-human system (software or hardware) interacting with the system of reference
	player*	activity participant not interacting with the system of reference (but often an actor with other systems)
	artifact, tool*	any artifact employed within an activity
	activity*	collection of actions or tasks undertaken for some purpose
	task, task case	action by an actor in interaction with the system of reference for some goal within an activity
	action*	action by a player for some goal within an activity

* new notation introduced for activity modeling

Source: Constaintine (2006)

References

Texts

Abrahamsson, Pekka; Warsta, Juhani; Siponen, Mikko T. & Ronkainen, Jussi. 2003. New directions on agile methods: a comparative analysis. *Proceedings of the 25th International Conference on Software Engineering*, pp. 244-254, held in Portland, Orgegon. IEEE Computer Society.

Abran, A.; A. Khelifi & W. Suryan. 2003. *Usability Meanings and Interpretations in ISO standards*. Netherland: Kluwer Academic Publishsers.

Aggarwal, K. K. & Singh, Yogesh. 2005. *Software engineering*. New Age International.

Alotaibi, Kholod. 2013. Improved requirements capture for usability enhancement through Usage-Centred Design based on a combined model-driven and agile approach. Mini-thesis submitted to University of Southampton, April, 2013.

Ambler, S. W. 2008. Agile adoption rate survey results. Ambysoft. Available at http://www.ambysoft.com/surveys/agileFebruary2008.html (accessed August 2012).

Amyot, D.; Logrippo, L.; Buhr, R. J. A. & Gray, T. 1999. Use case maps for the capture and validation of distributed systems requirements. *Proceedings of the IEEE International Symposium on Requirements Engineering*, held on 7-11 June, 1999, pp. 44-53.

Asnawi, Ani Liza; Gravell, Andrew M. & Wills, Gary B. 2010. An empirical study: Understanding factors and barriers for implementing agile methods in Malaysia. *5th International Doctoral Symposium on Empirical Software Engineering* (IDoESE), 15 September 2010, Bolzano-Bozen Italy.

Bachmann, Karen. 2004. *Usability requirements: making products successful*. Seascape Consulting, Inc.

Bak, Jakob Otkjaer; Nguyen, Kim; Risgaard, Peter & Stage, Jan. 2008. Obstacles to usability evaluation in practice: a survey of software development organizations. *Proceedings of the 5th Nordic Conference on Human-Computer Interaction: Building Bridges*, held in New York, pp. 23-32.

Barksdale, Jeremy T. & McCrickard, D. Scott. 2012. Software product innovation in agile usability teams: an analytical framework of social capital, network governance, and usability knowledge management. *International Journal of Agile and Extreme Software Development*, Vol. 1, No. 1.

Bashir, Salman & Qureshi, Rizwan Jameel. 2012. Hybrid software development approach for small to medium scale projects: RUP, XP & SCRUM. *Science International*, vol. 24, no. 4, pp. 381-384.

Baskerville, R.; Ramesh, B.; Levine, L. et al. 2003. Is 'Internet speed' software development different? *IEEE Software*, vol. 20, issue 6, pp. 70-77.

Beck, Kent. 2000. *Extreme programming explained: embrace change*. Addison-Wesley Professional.

Beckworth, G. & Garner, B. 1995. *An analysis of requirements engineering methods*.

Technical Report TRC9524. Deakin University, School of Computing and Mathematics.

Bennett. 2004. *Object-oriented systems analysis and design using UML*. Tata McGraw-Hill Education.

Bernhaupt, Regina. 2010. Human-centred software engineering. *Proceedings of the third International Conference*, HCSE 2010, held in Reykjavik, Iceland on 14-15 October, 2010.

Bevan, N. & Azuma, M. 1997. Quality in use: Incorporating human factors into the software engineering life cycle. *Proceedings of the 3rd IEEE International Software Engineering Standards Symposium and Forum*.

Blomkvist, S. 2005. Towards a model for bridging agile development and user-centered design. In Seffah, A.; Gulliksen, J. & Desmarais, M. C. (Ed's). *Human-centered software engineering – integrating usability in the development process*. Springer.

Boehm, Barry. 2002. Get ready for agile methods, with care. *Computer*, Vol. 35, No. 1, pp. 64-69.

Boehm, Barry; Port, Dan & Brown, Winsor. 2002. Balancing plan-driven and agile methods in software engineering project courses. *Computer Science Education*, vol. 12, no. 3, pp. 187-195.

Boehm, Barry. 2003. Value-based software engineering: reinventing. Newsletter, *ACM SIGSOFT Software Engineering Notes*, vol. 28, issue 2, p. 3.

Boehm, Barry W. & Turner, Richard, 2003. *Balancing agility and discipline: a guide for the perplexed*. Addison-Wesley Professional.

Boehm, Barry W. & Turner, Richard. 2005. Management challenges to implementing agile processes in traditional development organizations. *Software*, IEEE, Vol. 22, Issue 5.

Brooks, Laurence & Jones, Matthew. 1996. CSCW and requirements analysis: requirements as cooperation/requirements for cooperation. Computer Supported Cooperative Work, pp 10-20.

Brown, Derrick. 2008. The how to of essential modelling. IRM Training – White paper. Available at www.irm.com.au (accessed June 2012).

Bruning, Jens; Dittmar, Anke; Forbrig, Peter & Reichart, Daniel. 2008. Getting SW engineers on board: task modelling with activity diagrams. In Gulliksen, Jan & Harning, Morten Borup. 2007. *Engineering interactive systems: EIS 2007 Joint Working Conferences* EHCI 2007, DSV-IS 2007, HCSE 2007, held in Salamanca, Spain on 22-24 March, 2007.

Carmel, Erran; Whitaker, Randall & George, Joey F. 1992. Participatory Design versus Joint Application Design: trans-atlantic differences in systems development. In Muller, M. J.; Kuhn, S. & Meskill, J. A. *Proceedings of the Participatory Design Conference*, held in Cambridge MA, US, on 6-7 November, 1992.

Carroll, John M. 2000. Five reasons for scenario-based design. *Interacting with Computers*, vol. 13, pp. 43-60.

Carroll, John M.; Chin Jr., George & Koenemann, Jurgen. 1998. Requirements development in Scenario-Based Design. IEEE Transactions on Software Engineering, vol. 24, no. 12.

Chan, F. & Thong, J. 2008. Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems*, vol. 46, no. 4, pp. 803-814.

Chun, Andy Hon Wai. 2004. The agile teaching/learning methodology and its e-learning platform. *Lecture Notes in Computer Science*, vol. 3143, pp. 745-784.

Cohen, David; Lindvall, Mikael & Costa, Patricia. N.d. Agile software development. DACS State of the Art Practice Report. Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.201.2704 (accessed September 2012).

Cohn, Mike. 2004. *User stories applied: For agile software development*. Addison-Wesley Professional.

Comrey, A. L. & Lee, H. B. 1992. *A first course in factor anlaysis*. Hillsdale, NJ, Eribaum.

Constantine, Larry L. 2000. What do users want? Engineering usability into software. Constantine & Lockwood, Ltd.

Constantine, Larry L & Lockwood, Lucy A. D. 2001. Usage-centred engineering for web applications. *IEEE Software*, vol. 19, no. 2, pp. 42-50.

Constantine, Larry L & Lockwood, Lucy A. D. 2002b. Instructive interaction: making innovative interfaces self-teaching. Available at www.foruse.com/articles/instructive.pdf (accessed May 2012).

Constantine, Larry L. 2002. Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. *Information Age*, August/September issue.

Constantine, Larry. 2002b. Towards agility and software usability: toward lightweight usage-centred design. *Information Age*, August/September 2002 issue.

Constantine, Larry. 2003. Proceedings of DSV – IS'2003 – *10th International Workshop on Design, Specification and Verification of Inter-active Systems*, Lecture Notes in Computer Science, Berlin: Springer-Verlag.

Constantine, Larry; Biddle, Robert & Noble, James. 2003. Usage-centered design and software engineering: models for integration. *Proceedings of the 2003 International Conference on Software Engineering*, pp. 3-9.

Constantine, Larry & Windl, Helmut. 2003b. Usage-centred design: scalability and integration with software engineering. *The Second International Conference on Usage-Centered Design*, held on 18-22 October, 2003.

Constantine, Larry L. 2006. Activity modeling: toward a pragmatic integration of activity with Usage-Centred Design. Laboratory for Usage-centred Software Engineering.

Constantine, Larry L. 2011. *Software for use: A practical guide to the models and methods of usage-centred design*. Addison-Wesley Professional.

Costabile, M.F., De Marsico, M., Lanzilotti, R., Plantamura,V.L. & Roselli, T. 2005. On the Usability Evaluation of E-Learning Applications. *Proceedings of the 38th Hawaii International Conference on System Sciences* – 2005, ACMPress: New York, pp. 1-10.

Coughlan, Jane & Macredie, Robert D. 2002. Effective communication in requirements elicitation: A comparison of methodologies. Journal of Requirements Engineering, vol. 7, no. 2, pp. 47-60.

Cozby, P. C. 2003. *Methods in behavioral research with PowerWeb*. 8th edition. New York: McGraw-Hill.

Creswell & Clark. 2007. *Designing and conducting mixed methods research*. Thousand Oaks. Sage publications.

Czerwinski, Mary; Horvitz, Eric & Cutrell, Edward. 2001. Subjective duration assessment: An implicit probe for software usability. *Proceedings of IHM-HCI 2001*, Lille, France, September 2001, pp. 167-170.

Dabbagh, Nada. 2012. Performance support for advanced learning technologies selection and integration. *American Institute of Higher Education – The 7th International Conference*, held at Williamsburg, VA on 7-9 March, 2012, pp. 78-84.

Davey, Bill & Parker, Kekvin R. 2010. Technology in education: An agile systems approach. *Proceedings of the Informing Science & IT Education (InSITE) Conference* held in Southern Italy on 19-24 June.

Dubey, Sanjay Kumar; Anubha Gulati & Ajay Rana. (2012). Integrated model for software usability. *International Journal on Computer Science and Engineering*, vol. 4, no. 3, pp. 429-437.

Easterby-Smith, M. 1991. *Management research: An introduction*. London, Sage Publications.

Ferre, Xavier & Juristo, Natalia. 2001. Usability basics for software developers. *IEEE Software*, January/February issue, pp. 22-29.

Ferreira, Jennifer; Noble, James & Biddle, Robert. 2005. The semiotics of usage-centered design. *Eighth International Workshop on Organisational Semiotics*, held in Toulouse, France.

Field, A. 2009. *Discovering statistics using SPSS*. SAGE Publications.

Fitzgerald, B.; Hartnett, G. & Conboy, K. 2006. Customising agile methods to software practices at Intel Shannon, *European Journal of Information Systems*, Vol. 15, No. 2, pp. 200-213.

Folmer, Eelke; van Gurp, Jilles & Bosch, Jan. (2004). Scenario-based assessment of software architecture usability. *The IFIP Working Conference on Engineering for Human-Computer Interaction*, held in Hamburg, Germany, in July 2004, pp. 321-339.

Fowler, Martin. 2003. Using an agile software process with offshore development. ThoughtWorks. Available at http://www.martinfowler.com/articles/agileOffshore.html (accessed September, 2012).

Franke, David. 2006. Introduction to requirements gathering. Tyner Blain LLC. Available at tynerblain.com/.../20060606.IntroductionToRequirementsGathering.note (accessed April 2013).

Fuller, Anne. 2004. Usability/User interface design in agile processes. *Second OZCHI Conference*, held at Wollongong University, Australia, 21st-24th November, 2004.

Garson, D. 2012. *Testing statistical assumptions*. Statistical Associates Publishing.

Gonzalez, Maria Paula; Chesnevar, Carlos Ivan; Pinkwart, Niels & Lucero, Mauro J. Gomez. 2010. Developing argument assistant systems from a usability viewpoint.

Proceedings of the International Conference on Knowledge Management and Information Sharing, pp. 157-163.

Goransson, Bengt; Gulliksen, Jan & Boivie, Inger. 2003. The usability design process – integrating user-centered systems design in the software development process. *Software Process Improvement and Practice*, vol. 8, pp. 111-131.

Grady, Jeffrey O. 2010. System requirements analysis. Academic Press.

Gudykunst, W. B. & Ting-Toomey, S. 1988. *Culture and interpersonal communication*. Newbury Park, CA: Sage Publications.

Guerrero, F. & Eterovic, Y. 2004. Adopting the SW-CMM in a small IT organization. *Software*, Vol. 21, Issue 4, pp. 29-35. IEEE Computer Society.

Gulliksen, Jan & Boivie, Inger. 2001. Usability throughout the entire software development lifecycle: A summary of the INTERACT 2001 workshop. Technical Report 2001-2006, Department of Information Technology, Uppsala University.

Habrias, Henri & Frappier, Marc. 2010. Software specification methods. John Wiley & Sons.

Hair, J. F.; B. Black, Babin, Anderson & Tatham. 2007. *Multivariate data analysis*, 6th edition. Upper Saddle River, N. J., Pearson Prentice Hall.

Hall, E. 1983. The dance of life: the other dimension of time. New York: Doubleday. In Nishimura, Shoji; Nevgi, Anne & Tella, Seppo. 2008. Communication style and cultural features in high/low context communication cultures: a case study of Finland, Japan and India. *Proceedings of a Subject-Didactic Symposium in Helsinki*, part 2, held on 2nd February, 2008, pp. 783-796.

Hall, E. & Hall, M. 1990. *Understanding cultural differences: Germans, French and Americans*. Yarmouth: Intercultural Press.

Hammond, Judy; Gross, Tom & Wesson, Janet. (2002). *Usability: gaining a competitive edge*. Springer.

Harris, M.; Hevner, A. & Collins, R. 2009. Controls in flexible software development. *Communication of the Association for Information Systems*, Vol. 24, No. 43, pp. 757-776. Also in Hevner, 2010.

Harschnitz, Lesley. 2011. Gathering effective requirements. Golden Horseshoe SAS Users Group. ArcelorMittal. Available at www.sas.com/offices/NA/canada/...Fall.../Harschnitz-Requirements.pdf (accessed April 2013).

Haumer, Peter. 2004. Use case-based software development. Book chapter in Alexander, Ian & Maiden, Neil (Ed.). *Scenarios, stories, use cases: through the systems development life-cycle*. Wiley.

Hertzum, Morten. 2010. Images of usability. *International Journal of Human-Computer Interaction*, vol. 26, no. 6, p. 567-600.

Hevner, Alan. 2010. *Design research in information systems: theory and practice*. Springer.

Hirose, Michitaka. 2001. Human-computer interaction: INTERACT '01: IFIP TC: *13th International Conference on Human-Computer Interaction*, held on 9-13 July, 2001, in

Tokyo, Japan.

Hofstede. 2001. In Christine Falkenreck. 2009. *Reputation transfer to enter new B-to-B markets: Measuring and modelling approaches*, p. 59. Springer Science & Business Media.

Hollnagel, Erik. 2003. *Handbook of cognitive design*. CRC Press.

Hudson, William. 2001. Toward unified models in user-centered and object-oriented design. Chapter 9 in Harmelen, Mark Van. *Object modeling and user interface design*. Addison-Wesley.

Hussey, A.; MacColl, I. & Carrington, D. (2001). Assessing usability from formal user-interface designs. *Proceedings of Software Engineering Conference 2001*, held in Australia on 27-28 August, 2001, pp. 40-47.

Hwang, Wonil & Salvendy, Gavriel. 2010. Number of people required for usability evaluation: the 10±2 rule. *Communications of the ACM*, vol. 53, issue 5, pp. 130-133.

Ian, I.; Douglas, J. & Liu, Zhengjie. 2011. *Global usability*. Springer.

IEEE Std. 1061. 1992. *IEEE standard for a software quality metrics methodology*. IEEE Computer Society Press.

Jain, Piysh; Kumar, Sanjay & Rana, Ajay. 2012. Software usabilty evaluation method. *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 1, no. 2.

Johnson, R. B.; Onwuegbuzie, A. J. & Turner, L. A. 2007. Toward a definition of mixed methods research. *Journal of Mixed Methods Research*, vol. 1, no. 2, pp. 112-133.

Jokela, T. & Abrahamsson, P. 2004. Usability assessment of an Extreme Programming project: Close co-operation with the customer does not equal to good usability. PROFES 2004, *Lecture Notes in Computer Science*, Vol. 3009, pp. 393-407. Springer-Verlag.

Juristo, Natalia; Lopez, Marta; Moreno, Ana M. & Sanchez, M. Isabel. 2003. Improving software usability through architectural patterns. *ICSE 2003 International Conference on Software Engineering*, held in Portland, Oregon on May 3-11, 2003.

Kaiser, H. F. 1960. The application of electronic computers to factor analysis. *Educational and Psychological Measurement*, vol. 20, pp. 141-151.

Kane, David. 2003. Finding a place for discount usability engineering in agile development: Throwing down the gauntlet. *Proceedings of the Agile Development Conference 2003*.

Kautz, Karlheinz. 2009. Cultures of agility – Agile software development in practice. *20th Australasian Conference on Information Systems*, held on 2-4 December in Melbourne, Australia.

Kavitha, C. R. & Thomas, Sunitha Mary. 2011. Requirement gathering for small projects using agile methods. *IJCA Special on 'Computational Science – New Dimensions & Perspectives'*, NCCSE, 2011.

Kelly, Janet. 2014. Displacing use: exploring alternative relationships in a human-centred design process. *Design Studies*, vol. 35, issue 4, pp. 353-373.

Khalid, Asra; Zahra, Sobia & Khan, Muhammad Fahad. 2014. Suitability and contribution

of agile methods in mobile software development. *I. J. Modern Education and Computer Science*, vol. 2, pp. 56-62.

Kobbenes, Haakon & Folkman, Kristian. 2003. E-learning and usability: integrating the use and user dimenstion. *REN Report 2003: e-learning and usability*. San Francisco: Research and Educational Network.

Koskela, Juha & Abrahamsson, Pekka. 2004. On-site customer in an XP project: empirical results from a case study. *Software Process Improvement, Lecture Notes in Computer Science*, 3281, pp. 1-11.

Kothari, C. R. 2004. *Research methodology: methods and techniques*. New Age International.

Kruse, K. (2002). E-Learning and the Neglect of User Interface Design, E-LearningGuru.com.

Kulak, Daryl & Guiney, Eamonn. 2012. *Use cases: requirements in context*. Second edition. Addison-Wesley.

Lankhorst, Marc. 2012. *Agile service development: Combining adaptive methods and flexible solutions*. Springer Science & Business Media.

Larman, C. 2002. *Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process*. Second edition. Prentice Hall.

Larman, C. 2003. *Agile and iterative development: A manager's guide*. Addison-Wesley Professional.

Lauesen, Soren & Younessi, Houman. 1988. Six styles for usability requirements. *Proceedings of the REFSQ '98*. Presses Universitaires de Namur.

Lauesen, Soren. 2002. *Software requirements: styles and techniques*. Addison Wesley Publishing Company Incorporated.

Lee, Jason Chong; Judge, Tejinder K. & McCrickard, D. Scott. 2011. Evaluating eXtreme scenario-based design in a distributed agile team. *CHI 2011*, held on 7-12 May, in Vancouver, BC, Canada.

Lester, Cynthia Y. 2011. Combining agile methods and user-centered design to create a unique experience: An empirical inquiry. *The Fourth International Conference on Advances in Computer-Human Interactions*.

Lilburne, Ben; Devkota, Prajwol & Khan, Khaled Md. 2004. Innnovations Through Information Technology, *2004 IRMA International Conference*, held in New Orleans, USA.

Little, Todd. 2005. Context-adaptive agility: managing complexity and uncertainty. *Software*, IEEE, Vol. 22, Issue 3, pp. 28-35. IEEE Computer Society.

Macaulay, Linda. 1993. Requirements capture as a cooperative activity. In *Proceedings of the 1st IEEE International Symposium on Requirements Engineering*, held in San Diego, CA, on 4-6 January, 1993, pp. 174-181.

Macaulay, L. A. 1996. *Requirements engineering*. London: Springer-Verlag.

Manza, Ramesh R. 2010. *Computer vision and information technology: advances and applications*. I. K. International Pvt Ltd.

Marti, Patrizia & Bannon, Liam J. 2009. Exploring user-centred design in practice: some caveats. *Knowledge, Technology and Policy*, vol. 22, issue 1, pp. 7-15.

Mathews, T.; Judge, T. & Whittaker, S. 2012. How do designers and user experience professionals actually perceive and use personas? *Proceedings of the ACM Annual Conference on Human Factors in Computing Systems*, pp.1219-1228.

Maurer, Frank & Wells, Don. 2003. Extreme programming and agile methods – XP/Agile universe 2003: P*roceedings of the Third XP and second agile universe conference*, New Orleans, LA, USA, August 10-13, 2003.

McBreen, Pete. 1998. Using use cases for requirements capture. McBreen Consulting. Available at http://www.mcbreen.ab.ca (accessed May 2013).

McDonald, Malcolm & Wilson, Hugh. 2011. *Marketing plans: how to prepare them, how to use them*. John Wiley & Sons.

Memmel, T.; Bock, C. & Reiterer, H. 2007. Model-driven prototyping for corporate softwre specification. In Gulliksen, Jan. & Harning, Morten Borup. (Eds.). *Engineering interactive systems: EIS 2007 Joint Working Conferences EHCI 2007, DSV-IS 2007, HCSE 2007, Salamanca, Spain, March 22-24, 2007. Selected Papers.* Springer.

Moe, K. H.; Dwolatzky, B. & Olst, R. 2004. Designing a usable mobile application for field data collection. IEEE Africon 2004. *7th AFRICON Conference in Africa*, held on 15-17 September, 2004. Vol. 2, pp. 1187-1192.

Mueller, C. J. 2009. An economical approach to usability testing. *33rd Annual IEEE International Computer Software and Applications Conference 2009*, pp. 124-129.

Murphy, Gail C. 2010. Human-centric software engineering. *Proceedings of the FSE/SDP workshop on Future of Software Engineering Research*, held on 7-8 November, 2010 in Santa Fe, New Mexico, USA, pp. 251-254.

Nielsen, J. 1994. *Usability engineering*. Morgan Kaufmann series in Interactive Technologies. Morgan Kaufmann.

Nielsen, Jakob & Pernice, Kara. 2010. *Eyetracking web usability*. New Riders.

Nodder, C. & Nielsen, J. 2009. *Agile usability: Report on best practices for user experience on agile development projects*. 2nd edition. Nielsen Normal Group.

Norman, Donald A. 2005. Human-centred design considered harmful. *Interactions*, vol. 12, no. 4, pp. 14-19.

Novak, Marek; Binas, Miroslav & Jakab, Frantisek. 2010. Contribution of human-computer interaction in usability and effectiveness of e-learning systems. *8Th International Conference on Emerging eLearning Technologies and Applications*, held in the High Tatras in Slovakia on 28-29 October, 2012.

Nuseibeh, Bashar & Easterbrook, Steve. 2000. Requirements engineering: a roadmap. *Proceedings of the ICSE 2000 Conference on the Future of Software Engineering*, pp. 35-46.

Orr, Ken. 2002. *CMM versus agile development: religious wars and software development. Cutter Business Technology Council*, Vol. 3, No. 7.

Ovesen, Nis; Eriksen, Kaare & Tolestrup, Christian. 2011. Agile attitude: Review of agile

methods for use in design education. *International Conference on Engineering and Product Design Education*, held during 8-9 September 2011 at City University, London, UK.

Parsons, David & Lal, Ramesh. 2006. Hybrid agile development and software quality. Paper presented at the *14th International Conference on Software Quality Management*, in Southampton Solent University, UK.

Patton, J. 2002. *Hitting the target: Adding interaction design to agile software development*. OOPSLA 2002 Practitioners Reports. ACM Digital Library.

Patton, Jeff. 2003. Improving on agility: adding usage-centered design to a typical agile software development environment. ForUse 2003: *Proceedings of the Second International Conference on Usage-Centred Design*.

Patton, M. Q. 2002. *Qualitative research and evaluation methods*. London, Sage Publications.

Persson, John Stouby; Mathiassen & Aaen, Ivan. 2011. Agile distributed software development: enacting control through media and context . *Information Systems Journal*. Early view prior to publication, available at: http://onlinelibrary.wiley.com/doi/10.1111/j.1365-2575.2011.00390.x/abstract (accessed April, 2015).

Petriu, Dorin. 2002. Analysing software requirements specifications for performance. *Conference Proceedings of the 3rd International Workshop on Software and Performance*, pp. 1-9. Association of Computing, Mach., New York.

Pohl, Klaus. 2010. *Requirements Engineering: Fundamental, principles, and techniques*. Berlin, Heidelberg: Springer-Verlag.

Pohl, Klaus & Rupp, Chris. 2011. *Requirements engineering fundamental: a study guide for the certified professional for requirements engineering exam – foundation level – IREB compliant*. O'Reilly Media Inc.

Preece, Jenny. 2001. Sociability and usability in online communities: determining and measuring success. *Information Technology Journal*, vol. 20, no. 5, pp 347-356.

Preston, Carolyn C. & Colman, Andrew M. 2000. Optimal number of response categories in rating scales: reliability, validity, discriminating power, and respondent preferences. *Acta Psycholigica*, vol. 104, pp. 1-15.

Al-Rawas, Amer & Easterbrook, Steve. 1996. Communication problems in requirements engineering: a field study. In *Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering*, held at the Royal Society in London on 1-2 February, 1996.

Raza, Arif & Capretz, Luiz Fernando. 2010. Contributors preference in open source software usability: An empirical study. *International Journal of Software Engineering & Applications*, vol. 1, issue 2, pp. 45-64.

Rico, David F.; Sayani, Hasan H.; Sutherland, Jeffrey V. & Sone, Saya. 2009. *The business value of agile software methods: Maximising ROI with Just-in-Time processes and documentation*. J. Ross Publishing.

Rosenberg, Doug. & Scott, Kendall. 1999. *Use case driven object modeling with UML: a practical approach*. Upper Saddle River, NJ: Addison-Wesley Professional.

Rubin, Jeffrey & Chisnell, Dana. 2008. *Handbook of usability testing: How to plan, design and conduct effective tests*. Second edition. Wiley Publishing.

Sabharwal, Sangeeta. 2008. *Software engineering*. New Age International.

Satzinger, John W.; Jackson, Robert B. & Burd, Stephen D. 2008. *Systems analysis and design in a changing world*. Cengage Learning EMEA.

Saunders, M.; Lewis, P. & Thornhill, Adrian. (2009). *Research methods for business students*. Fifth edition. Prentice Hall.

Schneider, Jean-Guy & Johnston, Lorraine. 2005. eXtreme Programming—helpful or harmful in educating undergraduates? *Journal of Systems and Software*, vol. 74, issue 2, pp. 121-132.

Seffah, Ahmed & Metzker, Eduard. 2004. The obstacles and myths of usability in software engineering: Avoiding the usability pitfalls involved in managing the software development life cycle. *Communications of the ACM*, Vol. 47, No. 12.

Shilwant, S. & Haggarty, A. 2005. Usability Test-ing for E-Learning. Available at: http://www.clomedia.com/content/templates/clo_article.asp?articleid=1049 (accessed April 2013).

Silva, Fernando Selleri; Soares, Felipe Santana Furtado; Peres, Angela Lima et al. 2015. Using CMMI together with agile software development: A systematic review. *Information and Software Technology*, vol. 58, pp. 20-43.

Singh, M. 2008. U-SCRUM: An agile methodology for promoting usability. *Proceedings of the Agile 2008 Conference*, held in Washington, D.C. Institute of Electrical and Electronics Engineers.

Sofia. 2010. Software development process – activities and steps. Available at www.uacg.bg/filebank/acadstaff/userfiles/publ_bg_397_SDP_activities_and_steps.pdf (accessed June 2013).

Sommerville, Ian. 2007. *Software engineering: International computer science series*. 8th edition. Addison-Wesley.

Srivastava, Shirish C.; Chandra, Shalini & Lam, Hwee Ming. 2009. *Usability evaluation of e-learning systems*. IGI Global.

Stamelos, Ioannis G. & Sfetsos, Panagiotis. 2007. *Agile software development quality assurance*. Idea Group Inc.

Stanley, R. & Kurtz, P. 2011. Usability testing: A key component in e-learning design. *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2011*, held in Honolulu, Hawaii, USA, on 18 October, 2011.

Sundar, D. 2010. *Software engineering*. New Delhi, India: Laxmi Publications.

Sutton, D. C. 2000. Linguistic problems with requirements and knowledge elicitation. *Requirements Engineering Journal*, vol. 5, no. 2, pp. 114-124.

Tabachnick, B. G. & Linda S. Fidell. 2012. *Using multivariate statistics*. Pearson Education.

Tesar, Michael & Sieber, Stefanie. 2010. Managing blended learning scenarios by using

agile e-learning development. IADIS International Conference e-Learning. International Association for Development of the Information Society. Available at: http://www.iadisportal.org/digital-library/managing-blended-learning-scenarios-by-using-agile-e-learning-development (accessed April, 2015).

Tiwana, A. & Keil, M. 2004. The One-Minute Risk Assessment Tool. *Communications of the ACM*, Vol. 47, No. 11, pp. 73-77.

Tolfo, C. & Wazlawich, R. S. 2008. The influence of organizational culture on the adoption of extreme programming. *The Journal of Systems and Software*, vol. 8, pp. 1955-1967.

Turk, Daniel; France, Robert & Rumpe, Bernhard. 2005. Assumptions underlying agile software-development processes. *Journal of Database Management*, vol. 16, no. 4, pp. 62-87.

UsabilityNet. 2006. International standards for HCI and usability. Usability Net. Available at http://www.usabilitynet.org/tools/r_international.htm#9126-1 (accessed June 2012).

Venkatesh, V., Morris, M. G., Davis, G. B. & Davis, F. D. 2003. User Acceptance of Information Technology: Toward A Unified View. *MIS Quarterly*, vol. 27, no. 3, pp. 425-478.

VersionOne. 2008. The state of agile development 2008. VersionOne. Available at http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf (accessed August 2012).

Westfall, Linda. 2008. *The certified software quality engineer handbook*. ASQ Quality Press.

Winckler, Marco; Forbrig, Peter & Bemhaupt, Regina. 2012. Human-centred software engineering. *Proceedings of the 4th International Conference, HCSE 2012*, held in Toulouse, France, on 29-31 October, 2012.

Wurdel, Maik. 2011. *An integrated formal task specification method for smart environments*. Logos Verlag Berlin GmbH.

Wysocki, Robert K. 2011. *Effective project management: traditional, agile, extreme*. John Wiley & Sons.

Young, Ralph R. 2003. *Requirements Engineering Handbook*. Artech House Print.

Zave, P. 1997. Classification of research efforts in requirements engineering. *ACM Computing Surveys*, vol. 29, no. 4, pp. 315-321.

Zelenty, Valerie E. (Ed.). 1998. IEEE recommended practice for software requirements specification. Software Engineering Standards Committee. IEEE Std 830-1998. IEEE Computer Society.

Zhang, Tao. 2010. Complementary Classification Techniques based Personalized Software Requirements Retrieval with Semantic Ontology and User Feedback. *2010 IEEE 10th International Conference on Computer and Information Technology* (CIT), pp. 1358-1363.

Zhou, Xiao-Ying. 2009. Usage-Centered Design for Government Websites - A Practical Analysis to Canada Government Website. *Second International Conference on Information and Computing Science*, ICIC 2009, pp. 305-308.

Websites

Agile Manifesto http://www.agilemanifesto.org/principles.html

All About Agile http://www.allaboutagile.com/what-is-agile-10-key-principles

KwikSurveys site: http://kwiksurveys.com

Raosoft sample size calculator http://www.raosoft.com/samplesize.html

SUS http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

UsabilityNet: http://www.usabilitynet.org

Usability Professional Association: http://www.upassoc.org