International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:1, 2014

# Requirements Gathering for Improved Software Usability and the Potential for Usage-Centred Design

Kholod J. Alotaibi, Andrew M. Gravell

***Abstract*—**Usability is an important software quality that is often neglected at the design stage. Although methods exist to incorporate elements of usability engineering, there is a need for more balanced usability focused methods that can enhance the experience of software usability for users. In this regard, the potential for Usage-Centred Design is explored with respect to requirements gathering and is shown to lead to high software usability besides other benefits. It achieves this through its focus on usage, defining essential use cases, by conducting task modeling, encouraging user collaboration, refining requirements, and so on. The requirements gathering process in UgCD is described in detail.

***Keywords*—**Requirements gathering, Usability, Usage-Centred Design.

## I. Introduction

USABILITY is an important software quality attribute for ensuring software learnability, reliability, user satisfaction, etc. and for improved productivity [1]. For various reasons however, usability is often neglected in software design [2], and many methods, including pure agile methods, are incapable of ensuring usability while those that do attempt to incorporate elements of usability engineering as an additional component of design are also found to be inadequate [3]. It is established that the need arises for not only more usability focused methods, for which two examples are given, but methods that also provide for a more balanced approach between being plan-driven on one hand and agile on the other.

Usage-Centred Design (UgCD) is a software development methodology that incorporates a model-driven phase into an agile framework that is specially designed for ensuring high software usability [4]. The potential of UgCD for improved software usability is explored with respect to the important requirements gathering phase after first providing an overview of requirements gathering and the field of Requirements Engineering.

This paper thus introduces the concept of usability in software engineering, justifies its importance, details the inadequacy of many methods in ensuring usability, and outlines the need for more usability focused and balanced methods.

It then introduces the Usage-Centred Design (UgCD) software development methodology as having potential for leading to improved software usability focusing in the way it handles the gathering of usability requirements.

## II. Usability and Its Importance

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 [5]. It is recognised as a key software quality attribute[1], which also impinges upon other essential attributes such as reliability, efficiency, learnability, and overall satisfaction. These four sub-attributes comprise the classical attributes of usability to which more can be added, such as consistency and natural mapping [6], effectiveness and engaging [7], rememberability and few errors [8], etc. Any lacking in usability would be noticeable as it would cause dissatisfaction with the software [9], whereas ensuring a high degree of software usability, can lead, for example, to greater productivity [1].

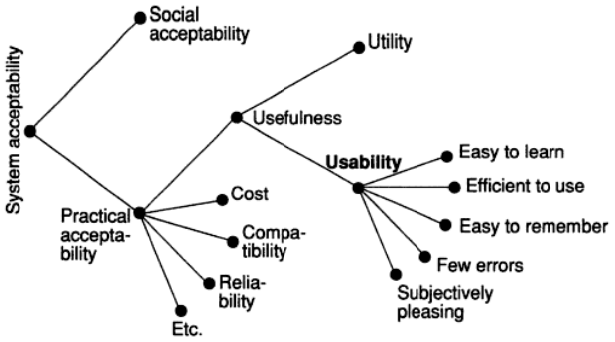


Fig. 1 Nielsen's five dimensional breakdown of usability [8]

As such, usability is a very important feature for software, yet often it is not given sufficient rigorous attention [10]. It is especially important, for example, for e-learning software as it helps to develop better systems with improved didactical and pedagogical approaches, but it is often neglected [2]. The lack of attention to ensuring usability can be partly explained by its resource intensive nature and requirement of a conducive developer mindset [11]. It therefore requires special training

Andrew M. Gravell is a PhD Supervisor and Academic staff in Electronic and Software Systems, Electronics and Computer Science, University of Southampton. (phone: +44 (0)23 8059 2741, e-mail: amg@ecs.soton.ac.uk).
Kholod J. Alotaibi is a PhD research student at the University of Southampton, UK's School of Electronics and Computer Science specialising in Software Usability Engineering, (e-mail: kja1g09@ecs.soton.ac.uk).

[1] ISO/IEC 9126-1

and close coordination. Also, usability testing can become tedious, least rewarding, and expensive [12], and in some cases, as with e-learning software, there is a lack of studies to show how usability can be evaluated [13]. Ian et al. [14] even reported an almost non-existent awareness of usability standards (among Malaysian developers).

## III. Inadequacy of Many Methods in Ensuring Usability

Pure agile methods lack the capability of ensuring software usability [15], as has been noted for Scrum [16] and XP [17]. Any claim to provide software having high usability is therefore questionable [18] because agile methods tend to simply ignore usability for end-users [19], and they use inadequate techniques for modeling tasks and users [20]. The original agile manifesto, which popularized agile methods, did not even allow for usability engineering, as it stated to favor 'individuals and interactions over processes and tools'. So unless additional design elements are incorporated, software developed using agile methods alone is not usually renowned for its usability.

Consequently, various attempts have been made to integrate elements into agile methods to ensure greater usability, such as methods based on User-Centred Design (UrCD), which is by nature user-centric. Incorporating elements of usability in an overall agile method is referred to as 'agile usability', and it usually involves elements of Usability Engineering or Interaction Design [4]. However, even many of these methods are inadequate at ensuring usability adequately because they are usually developed independently from traditional techniques established in the software engineering community that incorporate usability concerns more stringently [3].

## IV. Need for More Usability Focused and Balanced Methods

Accommodating for usability and detecting potential usability issues as early as possible is important because it can prevent costly late-cycle changes [21] among other benefits. Two notable attempts to adapt agile methods to deal with usability concerns are Scenario-based Design (SBD), and eXtreme Scenario-based Design (XSBD).

SBD was designed to specifically address certain technical challenges in software development [22] using a combined plan-driven and agile approach. SBD is interesting from the perspective of usability because it permits a range of possible usability consequences to be examined in the form of interdependencies and trade-offs. XSBD streamlines usability and development practices, involves usability testing, and also adopts practices to facilitate communication and information sharing [23]. However, whereas XSBD is able to focus on the most critical areas to ensure high-level goals are met, many other requirements tend to be uncovered too late. This suggests the need for more initial planning and earlier gathering of requirements.

An important aspect of usability focused software development is therefore to follow an approach that is able to take advantage of both plan-driven and agile approaches. Moreover, The actual method used should be adaptable according to the user requirements [24]. Despite an aversion to detailed planning in software development nowadays, especially among proponents of agile methods, some upfront planning provides three key potential advantages: (1) a reduction in uncertainty due to consideration of the likely outcomes and early adoption of corrective measures, (2) increase in understanding of the project goals and objectives, and (3) improvement in efficiency [25]. Consequently, a well-planned project is more likely to be of better quality, finish sooner, and cost less. Although this may be a 'painful' process to begin with, good planning pays off as a project progresses. On the other hand, spending too much time in planning poses a risk of 'high loss probability' due to the time that could have been spent more usefully getting on with the project itself [26], and it could also become a very expensive undertaking [27].

A more balanced approach can provide both essential qualities of control and discipline on one hand and agility or flexibility on the other [28]. The former is essential for dealing with highly specific requirements and constraints while the latter is desirable to allow for improvisation and adjustment. In addition to these technical aspects, social aspects of software development should also be considered in order to benefit from applying agile methods [29] through improved communication and collaboration. One such approach is introduced in the section that follows.

## V. Introduction to Usage-Centered Design

Usage-Centered Design (UgCD) is a software development methodology (developed by Constantine & Lockwood) that incorporates a model-driven phase in which requirements gathering is an important component, into an agile framework. It is specially designed for high software usability [4], [6], and has been demonstrated to do so [30]. It does this through ensuring the tasks users will have to carry out can be accomplished effectively and efficiently [31]. Thus, unlike with UrCD, the main focus is on usage rather than on users per se, hence the name UgCD. As acknowledged by Norman [32] who coined the term 'User-Centred Design', usage is more important than both the user and user interface.

UgCD makes use of essential use cases, which is based on Ivar Jacobson's concept of use cases in Object-Oriented Software Engineering, but “simplified and generalized down to the essential core of usage” [33]. Likewise, the modeling has its roots in Steve McMenamin and John Palmer's concept of 'essential modeling', which focuses on what the software is designed to do as an aid to conceptualizing the processes [34]. Modelling is characteristic of many traditional methodologies, but in UgCD, the models are designed to provide “conceptual and creative leverage for the least amount of effort on the part of analysts and designers” [35].

Besides the potential to enhance usability, UgCD can also cope well with scaling to larger projects [36], which is a problem for agile methods [37]. It also helps to make the

process of development run more smoothly [38], and guide decisions about the functionality to be incorporated into a design [39]. It is particularly effective for dealing with complex situations in which user performance is critical [35]. In short, UgCD provides 'a powerful punch' from its iterative process while delivering software of high quality that meets user expectations [40]. Table I highlights key characteristics of UgCD and how these differ from UrCD so as to make the distinctions clearer.

TABLE I
KEY DIFFERENCES BETWEEN URCD AND UGCD

| | User-Centred (UrCD) | Usage-Centred (UgCD) |
|---|---|---|
| Focus | Users (people) | Usages (activities) |
| Objectives | Improve user experience and satisfaction | Improve tools to support task accomplishment |
| Driven by | User input | Models |
| User involvement | Varied | Selective and collaborative |
| Specification | User descriptions and characteristics | User-system relationships |
| Design models | Realistic or representational | Abstract |
| Method of design | Iterative prototyping | Modelling and refinement |
| Processes /Design | Usually informal and varied | Fully specified and systematic |
| Coherence | Tends to be lower | Tends to be greater |

## VI. REQUIREMENTS GATHERING

The requirements gathering phase in the Software Development Lifecycle (SDLC) can play a major role in ensuring software usability. In practice, many developers tend to allocate insufficient time and effort to requirements gathering, such that requirements are usually either not documented at all or they get documented very late in the SDLC [41]. It is important however, as it provides the basis for the development work to follow [42]. Determining requirements prior to programming helps to better understand the software development project at hand, select an appropriate developmental method, and to satisfy the requirements [24]. The practice resulting in improved communication of requirements was shown recently to lead to a one-third drop in failed software projects [43]. Consequently, there is a growing renewed interest in the whole field of Requirements Engineering (RE) [44].

As for requirements gathering techniques, common methods include meetings, facilitated sessions, questionnaires, observations, document analysis and prototyping. A broad classification made by Nuseibeh & Easterbrook [45] is shown in Table II. IEEE recommendations in regard to gathering requirements specify that they should be correct, unambiguous, complete, consistent, ranked for importance/stability, verifiable, modifiable, and traceable [46]. Some of these characteristics are facilitated by documenting requirements while adhering to some set of standards, such as the use of graphical notations specified by the Unified Modelling Language (UML).

TABLE II
CLASSIFICATION OF REQUIREMENTS GATHERING TECHNIQUES

| 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 |

A traditional way of constructing models of requirements is using Jacobson's use case model, which involves identifying actors, use cases, and the system/sub-system boundary. Use cases are useful in that they highlight actor value linked with specific stakeholder goals [47], but as use cases are focused on a system's functionality, they are better suited only for documenting functional requirements [48]. The requirements themselves are only effective if they are complete, specific, measurable, achievable, connected, and signed off by clients [49].

Usually, the way requirements are gathered, is guided by an overall methodology or framework, of which there are many. Of interest in this paper are UgCD itself and others listed below that share some similar characteristics to UgCD to enable them to be compared.

*Activity-Centred Design* (ACD) – Activity modeling is used for systematic organization and contextual representation of tools with a focus on user activities and on tasks to be performed.

*Joint Application Design* (JAD) – A joint structured meeting is held which focuses on how the system will work in which user involvement is elicited using dynamic group techniques.

*Participatory Design* (PD) – Structured, facilitated interactions are held between designers and users.

*Scenario-Based Design* (SBD) – HCI scenarios are used to specifically overcome certain technical challenges.

*Usage-Centered Design* (UgCD) – Activity theory is applied with a methodological scaffolding involving task modeling and Just-in-Time Requirements (JITR) for further refinements with a focus on ensuring usability.

## VII. USABILITY REQUIREMENTS

Requirements can be divided into functional and non-functional, system and user, or conceptual and organizational, and so on. Another distinction of relevance herein, is between usability and non-usability requirements. Usability requirements are those 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” [48]. An

extensive breakdown of usability requirements was made by [50], in which they were shown to encompass conceptual, functional, non-functional, and also business requirements.

Identifying usability requirements helps to prioritize usability work and ensure usability goals are achieved, especially concerning efficiency and satisfaction [51], because usability requirements encapsulate usability goals [50]. Given this importance, Requirements Engineering (RE) cannot be ignored, especially in the development of customer-oriented systems [43]. An RE framework typically involves the three core activities of elicitation, documentation, and negotiation [52], and should be treated as a dynamic and cooperative process.

Usability is best assured when usability requirements are elicited at the beginning [44] in line with user goals and objectives, and when a usability assessment is undertaken at the conceptual stage of the SDLC [53]. Gathering requirements at this stage is important because this is the earliest point in time that potential performance issues can be addressed while the architecture is still being formulated [54]. The requirements should also be checked for correctness, completeness, and for being non-ambiguous and non-contradictory [55].

This plan-based approach is typical of traditional methods in software engineering. Pure agile methods on the other hand, are generally considered to be inadequate at handling requirements [56]. Their priority is for speedier and responsive software development [57], so they are better able to handle changing requirements instead due to their incremental development approach [58].

## VIII. Requirements Gathering under UgCD

UgCD adapts Jacobson's Use Case driven approach by conducting exploratory modeling of task cases to identify the roles and tasks and ascertain user requirements [59]. A user role is understood in UgCD 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” [60], and a role model contains a list of expected user roles described in terms of needs, interests, behaviors, responsibilities, etc.

The way requirements are gathered in UgCD takes the form of a dialogue in which there is mutual exploration and negotiation until a consensus is reached between the developers and users. Questions are raised and potential areas for investigation are identified based on information available at the time. This makes requirements gathering in UgCD a highly collaborative process due to the significant user involvement. In this way, UgCD is also distinguished from UrCD in that UgCD is not just user-centered, but also more user-involved. The collaboration is facilitated by the face-to-face meetings in a cooperative atmosphere, which are conversational rather than inquisitive, and organized around specific topic or aspects. Moreover, UgCD takes an 'essential' approach to gathering requirements in which the attention is on goals and objectives so as to ensure a focus on actual needs and requirements [60].

UgCD's ability to enhance software usability is due to its special consideration of usability arising from the gathering of requirements through task modeling [6]. Moreover, UgCD provides a very systematic way of establishing requirements, which particularly helps to prevent 'creep', which concerns expanding the system beyond the initial agreement, and 'leakage' of requirements, which refers to additional requirements that may arise and become part of the system without offering any benefits [60]. In short, UgCD by design ensures all those requirements are captured that help to accomplish each possible task, which is what ensures high software usability.

Table III summarizes key details of what takes place during the initial stages of the UgCD process with respect to modeling and requirements gathering. After the initial main meeting, any remaining omissions, irregularities, or ambiguities about the requirements are dealt with during the frequent subsequent meetings and continuous consultations that are characteristic of JITR. Notably, assistance and feedback is sought from users throughout the process at each stage.

TABLE III
Initial Stages of the Modeling and Requirements Gathering Process in UgCD

| Stage | Details of what takes place |
|---|---|
| 1. Pre-modelling | Identifying roles and tasks of users through defining essential use cases; discussion of purpose; requirements dialogue |
| 2. Model construction | Statement of purpose; Modelling tasks with a focus on satisfying software usability |
| 3. Model presentation | Model is presented and feedback is sought during the main meeting; Capturing of usability requirements |
| 4. Model refinement | Continuous consultations and further meetings to test and refine the requirements |

## IX. Summary and Plans for Evaluation

Usability is an important software quality for ensuring software can be easily learned, understood, used and made attractive and pleasing to its users, but it is often neglected at the design stage. This neglect is often due to the resource intensive nature of the challenge and tediousness of usability testing, but there is also a lack of studies in this area, especially in relation to e-learning software. Pure agile methods did not even allow for usability engineering, and whilst methods do exist to incorporate elements of usability engineering in both traditional and agile methods and promote a more user centered-design, they are usually found to be inadequate. The reason is that they are usually incorporated independently from traditional methods that specifically target usability.

There is a need for more balanced usability focused methods with initial planning and early gathering of requirements that can enhance the experience of software usability for users. Some upfront planning can provide the necessary control and discipline to deal with highly specific requirements by reducing uncertainty, increasing understanding of goals and objectives, and improving

efficiency. And, at the same time, some agility is necessary to allow for adjustments and refinements. The initial requirements gathering phase is therefore very important in software design for guiding the development of the software, particularly for capturing usability requirements for prioritizing usability concerns and ensuring high software usability.

In this regard, the potential for Usage-Centered Design (UgCD) was explored with respect to requirements gathering, which incorporates a model-driven phase into an overall agile framework. UgCD is shown to lead to high software usability besides other benefits such as scalability and suitability for user performance critical situations. It achieves this through its focus on usage over users per se, defining essential use cases, by conducting exploratory task modeling to identify roles and tasks, encouraging user-developer collaboration from the outset, engaging in mutual exploration and negotiation, allowing for refining requirements through continuous feedback, and so on. In short, UgCD is not just user-centred, but also more user involved, and it takes an 'essential' approach in gathering requirements by focusing on goals and objectives to ensure actual needs and requirements are met satisfactorily. The enhancement of software usability is a natural outcome of the special and systematic attention to usability concerns throughout the software development lifecycle.

In order to further establish the potential for UgCD, research has been planned to examine the existing requirements gathering process implemented for e-learning software development at a higher educational institution in Saudi Arabia. This focus on the initial requirements gathering phase contrasts with another study that also investigated UgCD's potential in the e-learning context [30], but which applied usability testing instead without attempting to recapture usability requirements. The UgCD approach will be applied in this study for specifically capturing usability requirements, and demonstrating how their capture is improved with respect to the attributes of completeness and preciseness. It is assumed that showing UgCD can 'better' capture usability requirements, can indicate its potential in creating more highly usable software.

## REFERENCES

[1] Xavier Ferre & Natalia Juristo. (2001). "Usability basics for software developers". *IEEE Software*, January/February issue, pp. 22-29.

[2] K. Kruse. (2002). "E-Learning and the Neglect of User Interface Design", E-LearningGuru.com.

[3] Ahmed Seffah & Eduard Metzker. (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

[4] Jeremy T. Barksdale & D. Scott McCrickard. (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.

[5] UsabilityNet. (2006). "International standards for HCI and usability". Usability Net. Available at http://www.usabilitynet.org/tools/r_international.htm#9126-1 (accessed August 2013).

[6] Natalia Juristo, Marta Lopez, Ana M. Moreno & M. Isabel Sanchez. (2003). "Improving software usability through architectural patterns". *ICSE 2003 International Conference on Software Engineering*, held in Portland, Oregon on May 3-11, 2003.

[7] Maria Paula Gonzalez, Carlos Ivan Chesnevar, Niels Pinkwart &, Mauro J. G. Lucero. (2010). "Developing argument assistant systems from a usability viewpoint". *Proceedings of the International Conference on Knowledge Management and Information Sharing*, pp. 157-163.

[8] J. Nielsen. (1994). "Usability engineering. Morgan Kaufmann series in Interactive Technologies". Morgan Kaufmann.

[9] Jeffrey Rubin & Dana Chisnell. (2008). "Handbook of usability testing: How to plan, design and conduct effective tests". Second edition. Wiley Publishing.

[10] C. Larman. (2002). "Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process". Second edition. Prentice Hall.

[11] Jakob Otkjaer Bak, Kim Nguyen, Peter Risgaard & Jan Stage. (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.

[12] C. J. Mueller. (2009). "An economical approach to usability testing". *33rd Annual IEEE International Computer Software and Applications Conference 2009*, pp. 124-129.

[13] Shirish C. Srivastava, Shalini Chandra & Hwee Ming Lam. (2009). *Usability evaluation of e-learning systems*. IGI Global.

[14] I. Ian, J. Douglas, & Zhengjie Liu. (2011). *Global usability*. Springer.

[15] David Kane. (2003). "Finding a place for discount usability engineering in agile development: Throwing down the gauntlet". *Proceedings of the Agile Development Conference 2003*.

[16] M. Singh. (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.

[17] T. Jokela & P. Abrahamsson. (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.

[18] Daniel Turk, Robert France & Bernhard Rumpe. (2005). "Assumptions underlying agile software-development processes". *Journal of Database Management*, vol. 16, no. 4, pp. 62-87.

[19] Larry L. Constantine. (2002). "Process Agility and Software Usability: Toward Lightweight Usage-Centered Design". *Information Age*, August/September issue.

[20] S. Blomkvist. (2005). Towards a model for bridging agile development and user-centered design. In A. Seffah, J. Gulliksen & M. C. Desmarais, (Ed's). *Human-centered software engineering – integrating usability in the development process.* Springer.

[21] T. Memmel, C. Bock & H. Reiterer. (2007). "Model-driven prototyping for corporate software 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.

[22] John M. Carroll. (2000). "Five reasons for scenario-based design". *Interacting with Computers*, vol. 13, pp. 43-60.

[23] Lee, Jason Chong Lee, Tejinder K. Judge & D. Scott McCrickard. (2011). "Evaluating eXtreme scenario-based design in a distributed agile team". *CHI 2011*, held on 7-12 May, in Vancouver, BC, Canada.

[24] Ian Sommerville. (2007). "Software engineering: International computer science series". 8th edition. Addison-Wesley.

[25] Robert K. Wysocki (2011). "Effective project management: traditional, agile, extreme". John Wiley & Sons.

[26] Barry Boehm. (2002). "Get ready for agile methods, with care". *Computer*, vol. 35, no. 1, pp. 64-69.

[27] F. Guerrero, & Y. Eterovic. (2004). "Adopting the SW-CMM in a small IT organization". *Software*, vol. 21, issue 4, pp. 29-35. IEEE Computer Society.

[28] Barry W. Boehm & Richard Turner. (2003). "Balancing agility and discipline: a guide for the perplexed". Addison-Wesley Professional.

[29] Ani Liza Asnawi, Andrew M. Gravell & Gary B. Wills. (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.

[30] Nada Dabbagh. (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.

[31] Larry Constantine, Robert Biddle & James Noble. (2003). "Usage-centered design and software engineering: models for integration." *Proceedings of the 2003 International Conference on Software Engineering*, pp. 3-9.
[32] Donald A. Norman. (2005). "Human-centred design considered harmful". *Interactions*, vol. 12, no. 4, pp. 14-19.
[33] Larry L. Constantine. (2000). *What do users want? Engineering usability into software*, p. 5. Constantine & Lockwood, Ltd.
[34] Derrick, Brown. (2008). "The how to of essential modelling. IRM Training – White paper". Available at www.irm.com.au (accessed June 2012).
[35] Larry Constantine. (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.
[36] Larry Constantine & Helmut Windl. (2003). Usage-centred design: scalability and integration with software engineering. *The Second International Conference on Usage-Centered Design*, held on 18-22 October, 2003.
[37] Jean-Guy Schneider & Lorraine Johnston. (2005). "eXtreme Programming—helpful or harmful in educating undergraduates?" *Journal of Systems and Software*, vol. 74, issue 2, pp. 121-132.
[38] Jeff Patton. (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*.
[39] Jennifer Ferreira, James Noble & Robert Biddle. (2005). "The semiotics of usage-centered design". *Eighth International Workshop on Organisational Semiotics*, held in Toulouse, France.
[40] J. Patton. (2002). "Hitting the target: Adding interaction design to agile software development". *OOPSLA 2002 Practitioners Reports*. ACM Digital Library.
[41] Michitaka Hirose. (2001). Human-computer interaction: INTERACT '01: IFIP TC: *13th International Conference on Human-Computer Interaction*, held on 9-13 July, 2001, in Tokyo, Japan.
[42] Ralph R. Young. (2003). "Requirements Engineering Handbook". Artech House Print.
[43] Klaus Pohl & Chris Rupp. (2011). "Requirements engineering fundamental: a study guide for the certified professional for requirements engineering exam – foundation level – IREB compliant". O'Reilly Media Inc.
[44] Tao Zhang. (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.
[45] Bashar Nuseibeh & Steve Easterbrook. (2000). "Requirements engineering: a roadmap". *Proceedings of the ICSE 2000 Conference on the Future of Software Engineering*, pp. 35-46.
[46] Valerie E. Zelenty (Ed.). (1998). "IEEE recommended practice for software requirements specification". Software Engineering Standards Committee. IEEE Std 830-1998. IEEE Computer Society.
[47] Peter Haumer. (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.
[48] Bennett. (2004). "Object-oriented systems analysis and design using UML", p. 121. Tata McGraw-Hill Education.
[49] Lesley Harschnitz. (2011). "Gathering effective requirements". Golden Horseshoe SAS Users Group. ArcelorMittal. Available at www.sas.com/offices/NA/canada/...Fall.../Harschnitz-Requirements.pdf .(2013accessed April )
[50] Ramesh R. Manza (2010). "Computer vision and information technology: advances and applications". I. K. International Pvt Ltd.
[51] Judy Hammond, Tom Gross, & Janet Wesson. (2002). "Usability: gaining a competitive edge". Springer.
[52] Klaus Pohl. (2010). "Requirements Engineering: Fundamental, principles, and techniques". Berlin, Heidelberg: Springer-Verlag.
[53] A. Hussey, I. MacColl, & D. Carrington. (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.
[54] Dorin Petriu. (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.
[55] 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).
[56] C. R. Kavitha & Sunitha Mary Thomas. (2011). "Requirement gathering for small projects using agile methods". *IJCA Special on 'Computational Science – New Dimensions & Perspectives'*, NCCSE, 2011.
[57] R. Baskerville, B. Ramesh & L. Levine et al. (2003). "Is 'Internet speed' software development different?" *IEEE Software*, vol. 20, issue 6, pp. 70-77.
[58] Barry W. Boehm & Richard Turner. (2005). "Management challenges to implementing agile processes in traditional development organizations". *Software*, IEEE, Vol. 22, Issue 5.
[59] Larry L Constantine & Lucy A. D. Lockwood. (2001). "Usage-centred engineering for web applications". *IEEE Software*, vol. 19, no. 2, pp. 42-50.
[60] Larry L. Constantine. (2011). "Software for use: A practical guide to the models and methods of usage-centred design", p. 79. Addison-Wesley Professional.

**Kholod Jeza Alotaibi** is a PhD research student at the University of Southampton, UK's School of Electronics and Computer Science specialising in Software Usability Engineering. She is currently engaged in a project titled 'Framework for Improved Capture of Usability Requirements through Usage-Centred Design'. This will involve soliciting the views and experiences of both developers and users of e-learning software at Saudi universities with the aim of showing ways to improve the software's usability. This research is under the supervision of Professor Andrew Gravell.

**Professor Andrew Gravell** is a senior member of the academic staff in Electronic and Software Systems at the University of Southampton, UK's School of Electronics and Computer Science. His interests include agile methods, e-administration, software development, and technology enhanced education.