Little Design Up-Front: A Design Science Approach to Integrating Usability into Agile Requirements Engineering

Sisira Adikari, Craig McDonald, and John Campbell

Faculty of Information Sciences and Engineering,
University of Canberra ACT 2601 Australia
{Sisira.Adikari,Craig.McDonald,John.Campbell}@canberra.edu.au

Abstract. In recent years, Design Science has gained wide recognition and acceptance as a formal research method in many disciplines including information systems. Design Science research in Human-Computer Interaction is not so abundant. HCI is a discipline primarily focusing on design, evaluation, and implementation where design plays the role as a process as well as an artefact. In this paper, we present a design science approach using "Little Design Up Front" to integrate the User-Centred Design perspective into Agile Requirements Engineering. We also present the results of two agile projects to validate the proposition that incorporating UCD perspective into Agile Software Development improves the design quality of software systems.

Keywords: Design Science, Agile Requirements Engineering, Usability.

1 Introduction

Usability has been identified as an important quality attribute of software products [1] but it has been classified as one of the Non-Functional Requirements (NFR) in Requirements Engineering (RE) [2]. A key aspect of traditional requirements engineering is to have formal requirements specified prior to software development. It also concentrates on functional requirements and ensuring that the developed products meet such requirements, rather than other NFR, which are considered less important [3]. The designation of usability as a less important NFR impacts the design because a reduced focus on user-centredness creates systems acceptance problems, necessitates rework and negatively impacts end user experience [4]. Current trends of software development increasingly favour agile development methods over plan-driven Software Engineering (SE) processes to better handle rapid change of stakeholder, business and technology requirements. Despite the success of Agile Software Development (ASD) reported by many software development organizations, none of the major agile development methods explicitly incorporate usability engineering practices in respective software development processes [5]. Recent research reported by Düchting et al. [6] involving two of the most popular agile models revealed that both had significant deficiencies in handling user-centered requirements. Accordingly, it is evident that ASD processes lack user-centric perspectives in their development methods and this likely to propagate usability

J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2009, LNCS 5610, pp. 549–558, 2009.

issues into finished products. As a result, end-user experience and satisfaction are directly affected.

In this paper, we present a design science approach using "Little Design Up Front" to integrate User-Centred Design (UCD) perspective into Agile Requirements Engineering. We also present the results of two agile software projects to validate the proposition that incorporating UCD perspective into ASD improves design quality of software systems.

2 Design Science

Design Science is a problem solving paradigm which aims at creating and evaluating innovative artifacts that address important and relevant organizational problems [7]. According to March and Smith, there are two fundamental design science processes: 'build' and 'evaluate', and four types of products namely: 'constructs', 'models', 'methods' and 'instantiations'. A construct forms the vocabulary of a domain, a model is a set of propositions expressing relationships among constructs, a method is a set of steps used to perform a task, and an instantiation is the realization of an artifact in its environment [8].

2.1 Design Science Research for Information Systems

In recent years, design science has gained a wide recognition and acceptance as a formal research method in many disciplines including Information Systems (IS). The Design Science paradigm has its roots in engineering and the sciences of the artificial [9]. Simon made the distinction between natural science and design science in that the former is concerned with how things *are* and the latter is concerned with how things *ought to be* [9]. Behavioral Science research is an origin of natural science and aims at developing and justifying theories which explain or predict organizational human phenomena surrounding the analysis, design, implementation, management, and use of information systems. On the other hand, Design Science Research (DSR) aims at creating innovations that define ideas, practices, technical capabilities, and product through the analysis, design, implementation, management, and use of information systems [7],[8]. As creating design solution artifacts for an important problem in Human-Computer Interaction (HCI) is a combined effort of both behavioral science and design science paradigms, these two research paradigms complement each other. Behavioral Science attempts to "understand" the problem. Design Science attempts to "solve" it. According to Iivari [10], design science is a contrast to natural-behavioral science research which aims at finding empirical regularities, whilst design science aims at building artifacts.

Hevner et al. [7] presented an IS research framework that combined both behavioral-science and design-science paradigms for understanding, executing, and evaluating IS research (see Figure 1). In the IS research framework, the *Environment* defines the scope of the problem domain that includes organizations, technology, and people. *IS Research* is the research effort conducted by applying behavioral science, through the use of theories that explain or justify the business problem, and design science to address the building and evaluation of artifacts designed to meet the identified business need. The *Knowledge Base* encompasses all the theoretical foundations, including the research methodologies and the kernel theories.

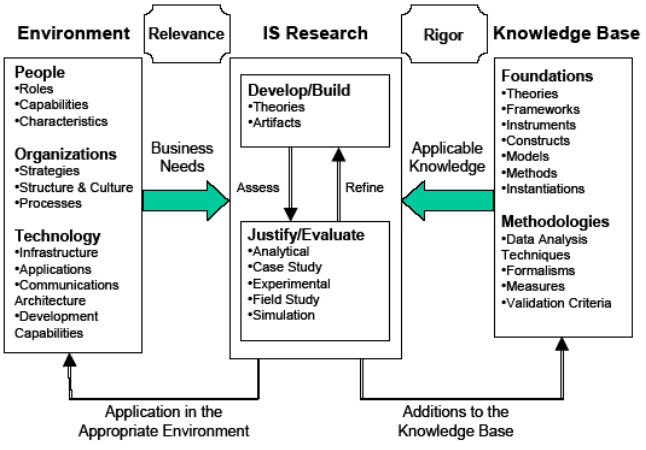

Fig. 1. Information Systems Research Framework [7]

In a recent paper, Hevner [11] further elaborated the IS research framework in terms of three inherent DSR cycles to enhance the understanding of high quality DSR in IS. Hevner pointed out that these three research cycles must be present and clearly identified in a DSR project.

These research cycles within the IS research framework are shown in Figure 2. According to Hevner, the *relevance cycle* connects the *contextual environment* of the research project with the *design science* activities. The main focus of relevance cycle is to capture problem to be addressed or requirements for the research and to provide design solution artifacts to the environment for study and evaluation in the application domain. The *rigor cycle* connects the *design science* activities with the *knowledge base* that informs the research project. That is, it ensures innovation by providing existing knowledge to the research. The knowledge base consists of foundations, existing experiences and expertise, and existing artifacts and processes. The main focus of rigor cycle is to provide applicable knowledge for design science activities

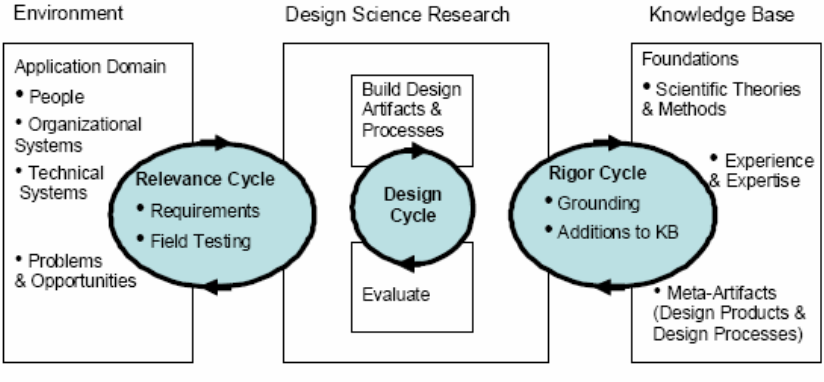

Fig. 2. Design Science Research Cycles [11]

and to feedback the updated knowledge to enrich the knowledge base. The internal design cycle iterates between core activities of building and evaluating the design artifacts and processes of the research. The main focus of the design cycle is to create, evaluate and refine design artifacts until a satisfactory design is achieved.

For this research project, we have deployed the information systems research framework associated with DSR cycles (Figure 1 and 2 above).

3 Agile Requirements Engineering and Practice

The main distinction between Agile Requirements Engineering (RE) and traditional RE is that the former welcomes rapidly changing requirements even late in the software development process and the latter gathers and specifies requirements up front prior to software development. The dynamic nature of most organizations makes continuously changing requirements normal, hence it is difficult to gather and specify complete, stable and accurate requirements up front. Rapid changes in competitive threats, stakeholder preferences, development technology, and time-to-market pressures make pre-specified requirements inappropriate [12].

A recent empirical case study [13] on ten software development organizations identified seven key agile RE practices namely: Face-to-face communication over written specifications, Iterative requirements engineering, Requirement prioritization, Managing requirements change through constant planning, Prototyping, Test-driven development, and Use review meetings and acceptance tests. These practices are in line with agile principles [14] such as: Satisfy the customer through early and continuous delivery of valuable software; Welcome changing requirements even late in development; Deliver working software frequently; Business and developers work collaboratively throughout the project; Build projects around motivated individuals; Face-to-face conversation as the most efficient and effective method of communication; Working software is the primary measure of progress; Promote sustainable development; Continuous attention to technical excellence and good design; Simplicity; Self-organizing teams and Regular reflections to become more effective.

4 User-Centred Design Integration with Software Engineering

In HCI literature, there are many user-centric methods and techniques that have been proposed to assist the production of usable, useful, and desirable software products [15], [16], [17]. Software product development still follows through a software development process where functionality is considered as the main priority. According to the literature, SE and HCI are largely two distinct communities. For the IEEE [18], SE is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software where as HCI is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use in a social context, and with the study of major phenomena surrounding them [19]. Importantly, HCI is by no means considered a central topic in SE and usability is considered as one of many non functional requirements and quality attributes [20].

As recently reported in the literature, there is a growing interest to incorporate user-centric perspective into SE practice so that usability awareness is widely known and software products become more user-centred and usable [21], [22]. This integrated approach is known as Human-Centred or User-Centred Software Engineering. Seffah et al. [23] discussed some of the most relevant HCI and SE integration frameworks and highlight their strengths and weaknesses as well as the level of objectivity in integrating HCI methods and principles for different software engineering methods. The frameworks they summarized were found to be useful for usability and software specialists who are interested in the development of methodologies and standards, who have researched or developed specific user-centered design techniques or who have worked with software development methodologies. Generally these frameworks provided insights in how to integrate user-centered best practices and user experiences with software engineering methodologies [20]. Discussing the importance of user modeling and usability modeling for user-centred software requirements, Adikari et al. [4] presented a framework for integrating ISO 13407 process model into a typical software development life cycle. The particular emphasis of the framework was its framework has the potential for defining the requirements to be more user-centred and task-oriented with lesser turnaround time.

5 Little Design Up-Front

Traditional RE stresses that requirements elicitation and specification required to be complete up front prior to the software development. Similar to traditional RE, UCD also assumes that contextual research and design will take place at the start of the project to provide detailed design information for subsequent development and evaluation. In agile environments, this assumption does not hold. Rather than defining requirements up front, agile software processes seek to follow an evolutionary approach to define requirements during the course of analysis, which is known as Just-In-Time (JIT) requirements analysis. As far as UCD is concerned, there should be at least a little contextual information available to support creating the design artifacts and proceed further. Therefore, JIT design approach is quite difficult and not appropriate for creating UCD focused artifacts in agile environments. As a practical solution, we propose Little Design Up Front (LDUF) - an approach providing *only required* details of UCD information *as needed* to support the analysis and design in agile iterations. The objective is to provide *only* sufficient LDUF information to support the popular agile JIT analysis and design so that UCD perspective can be considered without overloading existing agile practices. The LDUF is drawn from design solutions created in a DSR setting using environmental requirements, and applicable knowledge from the knowledge base as shown in Figure 3.

Figure 3 is similar to Figure 2 except that the *relevance* cycle in Figure 2 was replaced with Requirements (an input from environment to the DSR) and Solutions (an input from the DSR to the Environment) and these changes are in line with Figure 1 where Requirements and Solutions are represented by Business Needs and Application in the Appropriate Environment respectively. Moreover, the emphasis of *Create Little Design Artifacts* has been shown within DSR.

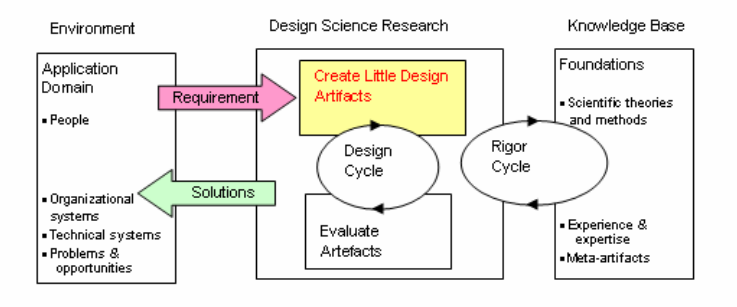

Fig. 3. Design Science Research Cycles with LDUF

6 Research Design

This research consisted of two agile projects. The first project was conducted as the baseline reference to compare the project incorporating user-centred design. The first project was a typical agile project with three iterations and its' research design is shown in Figure 4.

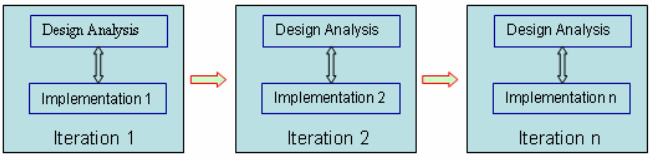

Fig. 4. Research design – Agile project 1

There were three defined roles in project 1 namely Product Owner, Agile Coach, and Agile Team. The product owner provided abstract level requirements for *both projects* and participated in tasks related to the product backlog analysis. The agile coach provided directions to the project and was responsible for removing any process impediments. The agile team made the necessary decisions to achieve goals of respective iteration and carried out the software development.

The second agile project was directed by a different agile coach and two user-centred designers worked with a *new agile team* in the design analysis providing the LDUF. The research design of the second agile project is shown in Figure 5.

6.1 Research Process

There were two different agile teams and agile coaches for project 1 and 2 and there were no other cross-over of resources excepting the product owner, who provided business requirements of an accommodation management system for both projects. The product owner was part of the each big team and was available in all iterations for requirements verification and validation. Project 1 ran first with three iterations. The first iteration was focused on requirements analysis and setting up the product

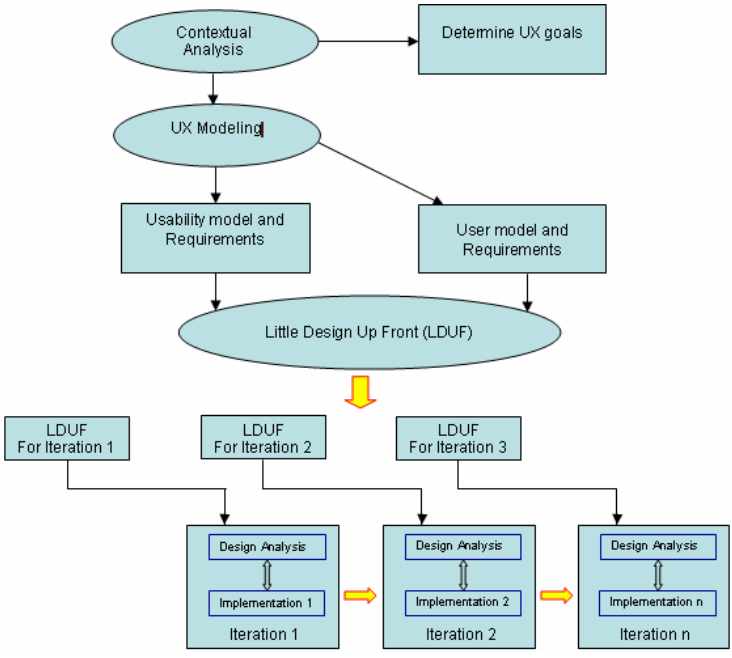

Fig. 5. Research design – Agile project 2

backlog. The agile team worked under the guidance and direction of the agile coach to produce working software. At the end of the first iteration, the agile team formally presented the first version of the working software to the product owner for assessment. In consultation and agreement with the product owner, the product backlog was then updated and the second iteration was planned. The second and third iterations were conducted in the same way as the first one based on similar agile settings and principles. At the end of the third iteration, the product owner formally assessed the final product delivered by the first project (P1) and signed off.

The second project was run in a similar fashion to the first project except that two user-centred designers were allowed to consistently engage with the team to put forward LDUF for design analysis. They worked very closely with the agile team and the product owner to create and assess paper-based artifacts in support of analysis, verification and validation. At the end of the third iteration, the product owner and user-centred designers formally assessed the final product delivered by the second project (P2) and signed off.

7 Product Evaluation

The product P1 and P2 were subjected to one-on-one usability evaluations with 16 participants who were randomly drawn from a large pool of users. The evaluation ran

in three stages. Firstly, the product P1 was evaluated with first 8 participants (group U1). Secondly, the product P2 evaluated with the second 8 participants (group U2) followed by first 8 participants (U1). Thirdly, the product P1 was evaluated with second 8 participants (U2). We followed this approach to minimize any learning effect bias in the assessments. We used a number of scenarios to guide the participant to go through the product and complete assigned user tasks.

After the evaluation, each participant was given a pack containing the Product Reaction Cards (PRC) [23] and System Usability Scale (SUS) [24] questionnaire. Participants were then asked to reefer to the PRC and tick all words that best described their user experience with the product and then to prioritize five of those words that they thought were most descriptive of the product. We then asked them to reason out why they chose those five words. We used Product Reaction Cards to aid participants to think deeply about their interaction experience. Finally the participant was requested to fill out the SUS questionnaire.

8 Results

A repeated measures Analysis of Variance (ANOVA) was conducted on the data for each question from the SUS questionnaire for both products. The aim was to determine if there was a significant difference of agreement of user groups in relation to their interaction with Product P1 and P2. Table 1 shows the mean response values for each product, statistical significance levels, the difference between mean values, and the percentage of change in mean values.

Table 1. Analysed results: Product P1 and P2

Question	P1 Mean	P2 Mean	P (Sig.)	P2 – P1	%
Q1. I think that I would like to use this system	2.81	3.44	.107	.63	22.4
Q2. I found the system unnecessarily complex.	2.69	1.94	.062	-.75	38.7
Q3. I thought the system was easy to use.	3.06	3.69	.042	.63	20.6
Q4. I think that I would need additional support to be able to use this system.	3.56	2.44	.011	-1.12	45.9
Q5. I found the various functions in this system were well integrated.	2.88	3	.662	.12	4.17
Q6. I found there was too much inconsistency in this system.	2.94	2.31	.149	-.63	27.3
Q7. I would imagine that most people would learn to use this system very quickly.	3.19	4.13	.042	.94	29.5
Q8. I found the system was very cumbersome to use.	2.88	2.31	.133	-.57	24.7
Q9. I felt very confident using the system.	3	3.56	.053	.56	18.7
Q10. I needed to think a lot before I could get going with the system.	2.88	2.19	.149	.69	31.5

According to the above Table, for each question, there is a positive difference of agreement from users for Product 2. Importantly, the agreements for the Q3, Q4, and Q7 are of significant difference (as the $P<0.05$ regarded as significance) yielding that Product 2 is easy to use (Q3), easy to learn (Q7) and product 1 requires additional support to be able to use (Q4).

Table 2 shows the SUS percentage for P1 and P2 reported by each participant. The mean of P1 = 47.31 and P2 = 52.95 and the difference is 5.61. The SUS usability difference of P1 and P2 is 11.92%. Accordingly product P2 found to be of better usability than product P1.

Table 2. SUS values for Product P1 and P2

Participant ID	SUS Usability %	
	P1	P2
U1	32.50	42.25
U2	42.50	45.00
U3	45.00	45.00
U4	45.00	50.00
U5	45.00	45.00
U6	45.00	45.00
U7	65.00	87.50
U8	40.00	42.50
U9	52.50	52.50
U10	52.50	52.50
U11	47.50	47.50
U12	55.00	65.00
U13	55.00	80.00
U14	42.00	45.00
U15	42.50	52.50
U16	50.00	50.00
Total	757	847.25
Mean	47.31	52.95
Difference %	**11.92**	

9 Conclusion

This paper presented the results of two agile projects to validate the proposition that incorporating a User-Centred Design perspective into Agile Software Development improves design quality of software systems. A design science approach using "Little Design Up Front" was used to integrate the User-Centred Design perspective into development process. The results show that users find products developed using this approach easier to learn, easier to use and require less support to be able to use.

Acknowledgements

We would like to thank Andrew Boyd, Donna Spencer, Dulan De Silva, Evan Laybourne, Narayanan Srinivasan, Rowan Bunning and Sandun Kodithuwakku for their advice/ support in this research project.

References

1. Jokela, T.: Guiding Designers to the World of Usability: Determining Usability Requirements through Team Work. In: Human-Centred Software Engineering – Integrating Usability in the Software Development Lifecycle, pp. 61–78 (2004)
2. Sommerville, I.: Software Engineering, p. 122. Pearson Addison-Wesley, England (2004)
3. Bevan, N.: Design for Usability. In: Proceedings of HCI International, pp. 762–767 (1999)
4. Adikari, S., McDonald, C., Lynch, N.: Design Science-Oriented Usability Modelling for Software Requirements. In: Proceedings of HCI International, pp. 373–382 (2007)
5. Kane, D.: Finding a Place for Discount Usability Engineering in Agile Development: Throwing Down the Gauntlet. In: Proceedings of the Agile Development Conference, pp. 40–46 (2003)

6. Düchting, M., Zimmermann, D., Karsten, N.L.: Incorporating User Centered Requirement Engineering into Agile Software Development. In: Proceedings of HCI International, pp. 58–67 (2007)
7. Hevner, A., March, S.T., Park, J., Ram, S.: Design Science Research in Information Systems. MIS Quarterly 28(1), 75–105 (2004)
8. March, S.T., Smith, G.F.: Design and Natural Science Research on Information Technology. Decision Support Systems 15(4), 251–266 (1995)
9. Simon, H.: The Sciences of the Artificial. MIT Press, Cambridge (1996)
10. Iivari, J.: A Paradigmatic Analysis of Information Systems as a Design Science. Scandinavian Journal of Information Systems 19(2), 39–64 (2007)
11. Hevner, A.: A Three Cycle View of Design Science Research. Scandinavian Journal of Information Systems 19(2), 87–92 (2007)
12. Merisalo-Rantanen, H., Tuunanen, T., Rossi, M.: Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases. Journal of Database Management 16(4), 41–61 (2005)
13. Cao, L., Ramesh, B.: Agile Requirements Engineering Practices: An Empirical Study. IEEE Software 25(1), 60–67 (2008)
14. Manifesto for Agile Software Development, `http://agilemanifesto.org/`
15. Nielsen, J.: Usability Engineering. Academic Press, San Diego (1993)
16. Mayhew, D.J.: The Usability Engineering Lifecycle. Morgan Kaufmann, San Francisco (1999)
17. Constantine, L.L., Lockwood, L.A.D.: Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. Addison-Wesley, Boston (1999)
18. IEEE: IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE, New York (1990)
19. ACM SIGCHI: Curriculum for Human-Computer Interaction. ACM Press, New York (1992)
20. Seffah, A., Desmarais, M.C., Metzker, E.: HCI, Usability and Software Engineering Integration: Present and Future. In: Human-Centered Software Engineering - Integrating Usability in the Software Development Lifecycle, vol. 8, Springer, Heidelberg (2005)
21. Zimmermann, D., Grötzbach, L.: A Requirement Engineering Approach to User Centered Design. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 360–369. Springer, Heidelberg (2007)
22. Seffah, A., Gulliksen, J., Desmarais, M.D. (eds.): Human-Centered Software Engineering - Integrating Usability in the Development Process. Springer, Heidelberg (2005)
23. Benedek, J., Miner, T.: Measuring Desirability: New Methods for Evaluating Desirability in a Usability Lab Setting. In: Proceedings of UPA, Oralando, Florida (2002)
24. Brook, J.: SUS: A Quick and Dirty Usability Scale. In: Jordan, P.W., McClelland, I.L., Thomas, B. (eds.) Usability Evaluation in Industry, pp. 18–195. Taylor and Francis, London (1996)