2007

Software Development Methodologies, Agile Development and Usability Engineering

David Parsons
Massey University,Auckland, d.p.parsons@massey.ac.nz

Ramesh Lal
Massey University,Auckland, r.lal@massey.ac.nz

Hokyoung Ryu
Massey University,Auckland, h.ryu}@massey.ac.nz

Manfred Lange
FirstData Corporation,Auckland, manfred.lange@peace.com

Software Development Methodologies, Agile Development and Usability Engineering

David Parsons, Ramesh Lal and Hokyoung Ryu
Institute of Information and Mathematical Sciences
Massey University
Auckland, New Zealand
Email: {d.p.parsons, r.lal,h.ryu}@massey.ac.nz

Manfred Lange
FirstData Corporation
Auckland, New Zealand
Email: manfred.lange@peace.com

Abstract

This paper examines the relationship between the practices of agile software development and usability engineering, and examines how these practices may be integrated within a single methodology. As agile methods have become increasingly popular, they have begun to replace structured approaches to software engineering. Usability engineering has historically tended to follow a development approach that is similar to traditional software engineering, rendering it apparently incompatible with agile methods. The focus of this paper is an analysis of the relationship between agile practices, on the one hand, and current practices of usability engineering on the other. We provide an experience report from an organisation where usability engineering has been integrated into an agile software development method. From this context we identify areas of compatibility and also areas of conflict, and suggest some strategies for agile development teams to incorporate key practices from usability engineering, while at the same time suggesting that usability engineers should embrace relevant agile techniques.

Keywords

Agile methods, usability engineering, software development

Introduction

In this paper we explore how agile software development methods (Fowler 2003) might be integrated with the usability engineering lifecycle (Mayhew 1999). We begin by comparing the core practices of both agile methods and usability engineering approaches. We then illustrate how both agile and usability engineering approaches may be integrated using a descriptive experience report. We conclude by discussing how our experience report may suggest strategies for agile development teams to ensure that usability is integrated into their own design processes.

Agile Methods

Past studies suggest that there is no single factor that contributes to failing to deliver a useful information system (Lyytinen 1987; Delone & McLean 1992; Jiang, Klein & Balloun 1998.) However, the main factors that negatively impact software development are related to the limitations and shortcomings of people, whether owners, developers or users (Ewusi-Mensah 1997; Charette 2005). Therefore a software development process should focus on managing interactions rather than technology; the soft skills of communication, collaboration and team work amongst the people involved in software development (Highsmith 2002). Modern software development has evolved from the mainframe world (centralised), through the open systems movement (decentralised), and into commodity systems (open source, self-organisation), while Business Process Management (formerly 'reengineering') tries to evolve companies into 'agile enterprises'. Thus there are two forces, technical and business, that require software development to be increasingly flexible and dynamic.

As a response to this changing environment, the mid 1990s saw the emergence of a new set of informal analysis and design approaches known as *lightweight methods*, later renamed *agile methods* and described by The Manifesto for Agile Software Development (Fowler 2003). Agile methods, by emphasising incrementally creating the software itself in close collaboration with the customer, are seen as an alternative to the more traditional emphasis on substantial planning, modelling, and artefacts. The Agile Manifesto states the following:

"we are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:

Individuals and interactions over processes and tools.

Working software over comprehensive documentation.

Customer collaboration over contract negotiation.

Responding to change over following a plan.

That is, while there is value in the items on the right, we value the items on the left more" (Highsmith 2002 pp. xvii).

Individuals and Interactions

A member of an agile team must attain and develop multiple technical skills in order to become a generalist rather than a specialist, though in practice we will still find that every team member has a strength (or preference) in some area, and that may be of benefit. Furthermore, if a particularly difficult problem arises, for example requiring specialist support with a particular software tool, external experts or support contract with vendors or service providers can be used. When an agile team is assembled for a project it does not require all the individuals in the team to have a very high level of technical skill or even similar levels and kinds of skill. It is expected that individuals will learn and master different technical skills by interacting with other team members when developing code, for example by co-location, pair programming and regular reflection workshops.

Working Software

Agilists firmly believe that it is "working software over comprehensive documentation" that shows the progress of a software development project. Having working software throughout a project means that it can be used for conferences, presales, or other customer related demonstrations. Nothing convinces more than running software, which demonstrates the trust that the company has in the quality of its product. No other artefacts can constitute this value (Highsmith 2002).

It is the incremental and iterative approach that enables agilists to deliver working software frequently and regularly for its customers. The shorter the iteration, the easier it is to deliver working software (Koch 2005). This approach has been complementary to a shift in the nature of software architectures away from large monolithic systems towards systems built from small software components, for example Service Oriented Architectures. Such systems enable smaller teams to work independently while collaborating on the common interfaces. Larger tightly coupled systems require a greater effort to coordinate the work across a larger team. In this way we can see that software architecture and methodology can interact and enable each other.

Customer Collaboration

An important premise of agile methods is that the exact requirements of the software product cannot be determined in advance, which is not the case in traditional software engineering approaches. All contracts include a section on 'change management', since the usual expectation is that the requirements will change during execution of the project. Therefore, the contract of an agile project negotiated now may not be applicable in future. For this reason agilists discount contract negotiation and emphasize collaboration among the stakeholders. Agilists believe that collaboration enables software to be developed quickly and delivered on a regular basis to the customers, helps to make decisions based on consensus, and assists learning and knowledge sharing among team members for improvement and quality purposes.

Responding to Change

Meticulously following a plan in an environment where business requirements change frequently will lead to implementation of an obsolete application. Therefore no software development project can be completely planned from the beginning. While agilists accept that minimal planning is important, they regard deviating from the plan as a result of new information as normal practice. The approach is to plan for short periods of time and plan often. An understanding of roles and responsibilities is also important in this planning cycle. Put simply: Customers (or customer proxies) are responsible for *what* gets implemented, e.g. features. The engineering team decide *how* things get implemented. This is not to say, however, that the conversation between them may not include these topics, for example a product manager may suggest the use of a specific technology.

Usability Engineering

According to Karat and Dayton (1995), "In most cases of the design and development of commercial software, usability is not dealt with at the same level as other aspects of software engineering." In contrast, usability

engineering does provide structured methods for achieving usability in the system development process, along with several basic disciplines including cognitive psychology, experimental psychology, ethnography and software engineering (Mayhew 1999). The usability engineering approach generally consists of several procedural phases, including requirements analysis, design, testing, development and installation. Much of this sequencing of phases is iterative should include every aspect of understanding the user and the tasks, contexts, and environment in which the application will be used, as suggested by user-centred design approaches (Sharp et al. 2007).

An essential feature of software development is that it is all about understanding how people use software in practice, and creating a mechanism to embody this descriptive aspect. Methodological approaches to usability engineering have been widely based on Human-Computer Interaction (HCI) literature. For example, an early reference to a usability engineering methodology was offered by Gould and Lewis (1985), who describe three major steps including eliciting user's tasks, testing, and iterative design. Since this pioneering article, others have added general frameworks for usability engineering. For instance, Button and Dourish (1996) describe how usability engineering has been evolving via three pivotal influences on software development; (i) adding cognitive scientists to design teams to provide psychological input on design; (ii) inserting usability enginnering methods and techniques with specific, written work documentation into the existing development process; (iii) redesigning the whole development process around usability engineering expertise, methods, and techniques.

Limitations of the Usability Engineering Model

From the previous description of usability engineering approaches, we can see that it closely resembles the traditional software engineering approach in its formality and insistence on up-front analysis and design (Vredenburg et al 2002). However the analysis of available literature on software development methodologies suggests that many of these have failed to incorporate the lessons of HCI research. For example the Capability Maturity Model Integration (Software Engineering Institute 2004), which assumes that good software development processes will lead to the development of usable software without the inclusion of any usability techniques or tools in the software engineering process.

Even if usability engineering is integrated into a software engineering approach, the usability aspects of that system will still suffer from the same problems as the methodology into which it is integrated, because of a lack of flexibility in requirements and also the prescriptive roles that it imposes. In a traditional software development environment we find architects, developers, testers and various other narrow roles, and the role of the user interface (UI) expert is cast in this traditional mould. However, from the agile perspective there are no distinct roles, rather different activities within the same role. One consequence of merging roles may be a significant reduction in specialist roles such as tester, architect and, of course, usability expert. Merging of roles also leads to a more democratic structure. For example the architect's role may be more of a consulting/coaching/mentoring model instead of imposing/mandating a specific architecture or design, giving more weight to the role of the software developers. The same would be true of usability experts.

Integrating Agile Methods and HCI concepts

If traditional software engineering methods have not always integrated usability engineering, the same may also be said for agile methods. Any use of usability engineering, or defining how much effort is put in to identify and fix any likely usability problems during an iteration, is not explicitly defined by agile approaches. In agile methods the requirements are stated as user stories or features, but it is not explicit which agile process would ensure the usability of a story or feature that would be implemented as a function.

The agile approach values delivery of something useful to the customer, who plays a vital role in identification, capture, and prioritizing of systems requirements. However, the individual who is the customer and is part of an agile development team may not actually be the real end user of the application being developed. Agile methods claim that the customer is in control throughout the development project, since the customer decides the requirement priorities for each iteration. However, in many agile projects the customer is a representative of the actual users, the sponsors of the project, and the management. The customer acts as a coordinator between them and the developers to provide the required business domain information to the developers, hence the end users do not take a direct part in software development. Worse, geographical or financial limitations may mean that a software development team member who may be a quality assurance person, an agile coach, a product manager, a software engineer or a marketing analyst may assume the role of 'customer'. Actual users may be involved by participating in review meetings or by giving them access to a prototype system, but the aspect of usability engineering is not necessarily taken into account. The major challenge, then, of an agile approach is how to identify the requirements of a system as accurately as possible from a customer who is not the actual end user and who will also lack the in-depth business knowledge that the actual users have. In this type of arrangement there is a possibility that accurate information is not being transferred between the developers and the actual users due to different interpretations of the information or the situation by the customer.

The agile culture of "minimal" design is another reason why agilists may see the usability engineering approach as being in direct conflict in the way they go about developing software. For example, there may be a requirement for an interface for entering an address that may need a field for the country. Initially this might be implemented using a simple text entry field. That might be completely sufficient for the early iterations of the screen. Later this would probably be replaced with a drop-down list. A further implementation might actually provide the user with a clickable map of the world. Alternatively, maybe entering an address is as easy as entering a Google Maps link. Some cell phones have GPS capabilities so sending a text message might enter the current address in a mobile UI, e.g. for calling a taxi. This simple example illustrates several different ways that a UI might be built, each being represented in a different increment. In contrast, usability engineering requires various upfront modelling and design alternatives before a best design is selected for implementation. It may also require building prototypes to convince the real users. The design, modelling, and prototyping are time consuming activities, which go through number of iterations based on feedback from the real users. While usability engineering identifies these as important artefacts which will help to develop a usable system, agilists believe that these artefacts do not deliver any value to the customer. With the agile approach the rule is to deliver a piece of working software at the end of each iteration period, so the real focus is on development and testing of code, and other activities are ignored as much as possible, including usability engineering activities such creating various designs, models and prototypes. The merging of roles in agile methods is also problematic, since the usability engineering approach assumes that specialised skills in user interface design are required to deliver a usable system. Depending on the emphasis placed on usability engineering by the customer, the company, and/or the engineering team, software developers in an agile setup may not necessarily pay attention to usability practices.

While the above issues show that agile methods have not explicitly integrated HCI concepts in their development processes, there are some studies that describe experiences of integrating HCI practices with an agile approach (Rittenbruch et al 2002; Hansson, Dittrich & Randall 2004). There are several similar approaches and processes between HCI and agile methods, in particular eXtreme Programming (XP) (Sharp, Robinson & Segal 2004), and similarities between agile methods and a user-participatory design approach, suggesting we can combine the culture, education, techniques and tools of HCI and agile methods to design usable software (Sharp et al 2006). The participatory and iterative approach of usability engineering is similar to the agile approach, where iterative development is a critical process for delivering software quickly and regularly to the customer, and the collaborative aspects of agile methods integrate well with user-participatory design practices (Bodker 2002). Therefore there should be a common core of compatibility that it may be possible develop. It may also be the case that each approach can benefit from the other. According to Sharp et al. (2006), agile methods lack guidance on requirements elicitation, user interaction, and user interface and they point out that an agile customer equipped with a user-participatory design approach should be able to more comfortably guide the agile development process. Bellotti et al (2002) suggest that a failed project based on user centred design was followed by a successful attempt using XP. However one might draw different conclusions from their paper. First, that the failure of the usability centred engineering project was mainly due to factors other than the chosen methodology, while the success of the XP project could in some respects be seen as significant due to its inclusion of usability engineering practices. Because the developers had begun with a usability engineering approach, many of their original practices influenced the way they applied XP, such that usability became a key issue in user testing.

A lesson that we might draw from these studies is that software development processes can be improved by building on the best practices of different approaches. However we should also be aware that most agile methods are not just a set of best practices (that is, a bullet list of items), but instead a system of practices that support/facilitate each other and that mitigate any shortcomings of individual techniques. For example, simple design requires refactoring, as does minimum upfront design. Refactoring, in turn, requires a test-driven approach, and so on. The agile approach expands on the individual specialised skills and plan-based effort of software engineering and life cycle methods to include team effort, combined with the multiple technical skills of an individual developer. If we wish to integrate aspects of usability engineering into an agile approach, we need to recognise the points of synergy so that integration is meaningful and beneficial. For example, one important contribution of integrating a user-participatory design approach into an agile method is that it mandates a process that involves real users, which most agile methods lack. The user-participatory design practices of contextual inquiry and design suggestion, design evaluation, and usability testing by real users, enable software development teams to implement systems that are useful to those actual users. In the next section we briefly describe an experience report on how aspects of user-participatory design have been integrated into an agile approach.

Experience Report

UtilSoft is a division of a large business process outsourcing provider located in the US. Part of UtilSoft's offerings require the development of software, which usually includes the implementation of a user interface

(UI) as well. Generally, the development methodology used is a combination of techniques from Scrum and XP, most of them adapted to the individual needs of each software engineering team. This experience report briefly describes the major HCI activities that are used as part of development and how they are integrated into the overall process.

Because some systems are used by customers who have thousands of users, UtilSoft regard user-participatory design as so important that they have made it an integrated part of their development approach. For instance, an important metric is the average call handling time of a customer service representative. An improved UI could perhaps reduce the average call handling time by around 1 second. This may appear to be a small improvement, given an average call handling time of, say, 300 seconds[1], but this can result in savings worth one full time employee's annual salary for some customers. This example illustrates the impact of usability from an operational perspective, but on the sales and marketing side a UI that "sells" is an important factor for the company's success, too.

The primary means of ensuring a high HCI quality in the agile software development process is to have a user interface expert co-located with the development team. One such expert can work with multiple teams. This cross-team role also has a business and technical background as the artefacts created by different teams need to be tightly integrated. As a consequence the UI needs to be as consistent as possible across different parts of the resulting solution. The UI expert's tasks include the basic layout of the UI, and providing user interface design guidelines. These include recommendations for how to use particular components for any given type of interaction (e.g. where and how to use a drop-down list, radio-buttons, panels, etc.), but also behavioural aspects of the UI, e.g. in which cases is it acceptable to submit the entire page back to the web server, and in which cases an Ajax (Asynchronous JavaScript and XML) enabled partial page refresh may be the better solution. The UI expert has also a broad domain knowledge, which allows him/her to always assess UI guidance from a business perspective. Furthermore, the UI expert works with sales and marketing to ensure consistency with corporate standards. In some cases local regulations require a deviation from the standard functionality, for example when an external credit score may only be displayed to users with special permissions.

The software development process is basically controlled through the use of backlogs. Each backlog is owned by a product manager, who in UtilSoft's case also serves as the customer proxy. The product manager is responsible for providing stories for the backlog, and prioritising these stories following estimation by the engineering team. A story usually describes a scenario of how a user interacts with the system. Typically the team starts with simple stories first and then incrementally increases the functional footprint by expanding the initial story with subsequent stories, which cover variations or additional aspects. Very often, stories cover a 'screen flow' sequence, and the product manager conceives an initial picture (metaphorically, not literally) of screens and flows. In collaboration with the engineering team, but also with the UI expert, the product manager's understanding evolves into a consistent vision of what is desirable and what is economically reasonable. Sometimes there are technical reasons why certain design solutions are not possible, although the engineering team is constantly pushing the envelope of what is achievable.

A third element towards an excellent user experience is the regular assessment by an external consulting company which specialises in HCI. Such external consultants bring with them experience from working with other clients and awareness of the latest research in the field. At least once a year, sometimes more often, the consultant's experts spend an entire week assessing different parts of the application. The findings usually contain a list of recommendations and guidelines with regard to further improvements in the user experience. These guidelines are then reflected back into the stories and also influence how the engineering team implements the system.

There are also technical aspects with regards to HCI. Most of UtilSoft's customers require customization of the user front end. Therefore the systems are developed in a modular and layered way allowing for a highly customizable solution. On the UI this means the use of Cascading Style Sheets (CSS), which to a large degree allow the 'skinning' of the screens. This represents an example of a per-customer customization. UtilSoft is also evaluating technologies enabling per-user customization. Among these one might find technologies such as portlets.

The bottom line is that the combination of close collaboration of co-located product manager, co-located UI expert, and the engineering team, and of supportive design and technology choices is the 'secret' of a successful integration of HCI activities into an agile development approach. Note that the integrated cross-functional approach includes not only HCI but also other aspects such as performance and scalability, which are ensured at the same time using appropriate techniques. It all comes together successfully if the right balance between all the different objectives is chosen. Table 1 summarises some of the key areas where our experience report indicates

[1] 300 seconds is just example for illustrative purposes. The actual numbers vary from customer to customer and are typically company confidential.

that usability engineering and agile methods may be successfully combined. In some cases we are indicating that usability engineering needs to adapt to the agile context, for example by using a more iterative approach, and by testing throughout the project lifecycle. In other cases we are seeing aspects of usability engineering informing the activities of agile projects, for example by incorporating user scenarios and including UI specialists in the team.

Table 1: Integrating aspects of usability engineering and agile methods

Usability Engineering	Combined Approach	Agile Methods
Waterfall type approach, with analysis, design, implementation and deployment phases, but iteration within/across phases	Use Iterative development throughout	Iterative approach based on working software
Documenting user scenarios (or user profiles and personas) in the analysis phase	Merge user scenarios (or user profiles and personas) with stories throughout the cycle	Requirements encapsulated in user stories
Specify full design (or partial prototypes) in design phase	Allow working prototype to emerge	Design emerges
Begin design with low fidelity prototypes including paper prototypes	Use software for prototypes	Prototypes are architectural and embedded in code
Assemble a multidisciplinary team to ensure complete expertise	Ensure team includes UI expertise among other skills	Business people and developers must work together daily throughout the project
Use surveys, field studies and usability testing in the testing phase	Test the UI in the user context regularly, throughout all phases	Drive development with testing and integration
Requires usability specialists	Use usability specialists across multiple projects, either in house or by using external consultants or contractors	Requires generalists

Conclusion

Evolving from more structured approaches to software engineering, agile methods are a flexible approach for software development where development teams continuously strive for improvement in their processes based on reflective practice. However, agile methods have not typically incorporated HCI and usability techniques and tools into their software development processes. Incorporating HCI and usability processes is important, because usability of the software product is an essential quality. In this paper we have explored the key features of agile methods and usability engineering, reviewed some examples from the literature where these two aspects of development have been used together, and provide some data from an experience report where usability engineering has been integrated into an agile development processes. It is clear that, while the traditional approach to usability engineering is incompatible with agile methods, it is possible to integrate certain practices from usability engineering into an agile approach. It is clear that the level of research carried out so far into the integration of usability engineering and agile methods is relatively limited. However, from both the existing literature and our own experience report it appears that there are many benefits that can be gained from combining these approaches. Nevertheless, further field studies are required to identify in greater detail aspects of best practice and the benefits that can result.

References

Bellotti, V, Ducheneaut, N, Howard, M, Smith, I & Neuwirth, C 200, 'Innovation in extremis: evolving an application for the critical work of email and information management', paper presented to Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, London, England, 2002.

Bodker, S, Iversen, J 2002, 'Staging a professional participatory design practice: moving PD beyond the initial fascination of user involvement', paper presented to the Second Nordic Conference on Human-Computer Interaction, pp. 11-18, Aarhus, Denmark

Charette, R 2005, 'Why Software Fails', *IEEE Spectrum*, no. 42, pp. 42-49.

Delone, W & McLean, E 1992, 'Information Systems Success: The Quest For the Dependent Variable', *Information Systems Research*, vol. 3, no. 1, pp. 60-95.

Button, G, Dourish, P 1996, 'Technomethodology: paradoxes and possibilities', paper presented to the Conference on Computer-Human Interaction, pp. 19-26

Ewusi-Mensah, K1997, 'Critical Issues in Abandoned Information Systems Development Projects', *Communications of the ACM*, vol. 40, no. 9, pp. 74-80.

Fowler, M 2003 *The New Methodology*, viewed 5 June 2007 <http://www.martinfowler.com/articles/newMethodology.html>.

Gould, J, Lewis, C 1985, 'Designing for usability: Key principles and what desingers think', *Communications of the ACM* 28 (3), pp. 360-411

Hansson, C, Dittrich, Y & Randall, D 2004, 'Agile processes enhancing user participants for small providers of off-the-shelf software', paper presented to Fifth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP2004), Garmisch-Partenkirchen, Germany, June 6-10.

Highsmith, J 2002, *Agile Software Development Ecosystem*, Boston, Addison-Wesley.

Jiang, J, Klein G, & Balloun, J 1998, 'Perceptions of System Development Failures', *Information and Software Technology*, vol. 39, pp. 933-937.

Karat, J, Dayton, T 1995 'Practical education for improving software usability', paper presented to the conference on Computer-Human Interaction, pp. 162-169.

Koch, A 2005, *Agile Software Development: Evaluating The Methods For Your Organization*, Boston, Artech House.

Lyytinen, K 1987, 'Different Perspectives on Information Systems: Problems and Solutions', *ACM Computing Surveys*, vol. 19, no. 1, pp. 5-46.

Mayhew, D 1999, *The Usability Engineering Lifecycle*, Morgan Kaufmann

Rittenbruch, M, McEwan, G, Ward, N, Mansfield, T & Bartenstein, D 2002, 'Extreme Participation: Moving Extreme Programming Towards Participatory Design', paper presented to Seventh Biennial Participatory Design Conference, Malmo, Sweden, June.

Software Engineering Institute 2007, *Capability Maturity Model Integration* viewed 5 June 2007 <http://www.sei.cmu.edu/cmmi/>.

Sharp, H, Robinson, H, & Segal, J 2004, 'eXtreme Programming and User-Centred Design', paper presented to 18th British HCI Group Annual Conference: HCI2004 Design for Life, Leeds Metropolitan University, UK.

Sharp, H, Biddle, R, Gray, P, Miller, L & Patton, J 2006, 'Agile Development: Opportunity or Fad', paper presented to Conference on Human Factors in Computing Systems (CHI 2006), pp. 32-35, Montreal, Canada.

Sharp, H, Rogers, Y, Preece, J 2007 *Interaction Design: Beyond Human-Computer Interaction* (2nd edition), Wiley.

Vredenburg, K, Isensee, S, Righi, C 2002, *User-Centered Design: An Integrated Approach*, Prentice Hall

Copyright