Examining the Foundations of Agile Usability with eXtreme Scenario-based Design

Jason Chong Lee, D. Scott McCrickard
Center for Human-Computer Interaction,
Department of Computer Science, Virginia Tech
Blacksburg VA, USA
{chonglee, mccricks}@vt.edu

K. Todd Stevens
Meridium, Inc.
Roanoke, VA, USA
tstevens@meridium.com

Abstract— **The increasing use of agile methods to develop UI-intensive systems has led to a need to find ways of integrating usability into agile teams—reconciling the convergence and divergent points between the two areas. Agile usability researchers at Virginia Tech have partnered with Meridium, Inc. to develop and implement an integrated approach known as eXtreme Scenario-based Design (XSBD). Based on an analysis of core values and principles of both areas, and work from other agile usability researchers we identified four requirements that need to be met for an integrated approach to work effectively. We report on the results of using XSBD to develop a product at Meridium, summarizing how it addresses those requirements and draw conclusions about the effectiveness of the approach—making connections back to core principles of agile usability.**

Keywords-agile, usability, extreme scenario-based design, central design record

I. INTRODUCTION

Agile methods are being used in an ever growing number of contexts by more companies as a way to mitigate many risks of software development. A number of these methodologies, such as extreme programming (XP) and Scrum, were created by practitioners in industry [2][23] and focused on empowering software developers to continuously deliver software that meet customer needs. As a result, these methods did not include much guidance for how to develop usable software. This was problematic for UI-intensive systems as the consequences of poor usability include reduced productivity, lost sales, lower user satisfaction and danger to human life for safety-critical systems. This oversight was due to a number of reasons. Many early agile projects involved relatively simple user interfaces. In addition, oftentimes there was no need to differentiate between customers and the end users as they were often the same group. This may have led to the expectation that close collaboration with the customer would ultimately lead to a usable end product [11][14][21]. As agile methods have been used to develop more UI-intensive systems, practitioners have begun to address this shortcoming using a variety of techniques [4][5][16][19][20][21].

There are open questions about how to best integrate usability into agile teams for projects where usability is a key quality concern while accounting for issues such as differing goals, practices, and mindsets. Our past work on integrating usability and agile methods, in addition to the other active research in agile usability methods, has uncovered a number of convergence and divergence points between agile methods and usability methods.

In this work, we explore the foundations of agile and usability engineering methods to better understand the commonalities and differences between the two areas and how they can be integrated and put into practice. Based on this information and past experiences with implementing usability into agile teams, Lee and McCrickard developed a specific instantiation of an agile usability approach known as eXtreme Scenario-based Design (XSBD) at Virginia Tech [12][13]. They partnered with Meridium, Inc., a leading software and services company in the asset performance management domain, to develop and implement this XSBD process into their organization. We present the results from the use of the XSBD process in one of their development teams for a system where usability was a key concern—making explicit how it can help address many of the tension points between agility and usability. We then make connections back to key principles and values that an agile usability team should adhere to—building on the existing values and principles as stated in the agile manifesto [3].

II. BACKGROUND

In this section, we look at the values and principles of the agile manifesto and look at the convergence and divergence points between agile and usability. We look at how they are addressed by different integration methods.

Agile methods value *individuals and interactions over processes and tools* [3]. Similarly, usability methods emphasize the importance of working with stakeholders using participatory design, ethnographic methods and usability testing [6][10][22]. Agile methods also value *customer collaboration over contract negotiation.* From a usability standpoint, the problem with this statement is the absence of any mention of the importance of end user collaboration. In addition, the agile values of *working software over comprehensive documentation a*nd *responding to change over following a plan* are somewhat at odds with usability approaches such as scenario-based design which follow a layered approach where the entire

user interface is designed in phases of increasing fidelity—rather than developed in slices where working parts of the system are made in increments [22]. This is because one problem with incremental development is that the resulting user interface can be disjoint, piecemeal and lack a cohesive overall vision [18]. Addressing this problem is one of the focus points of this work. Regardless, both agile and usability methods use cyclical development processes so both areas acknowledge that requirements cannot be fully defined at the start of a project and may change throughout it.

One key agile principle is that *business people and developers must work together daily throughout an agile project* [3]. However, many usability approaches assume end to end involvement of usability people in the development process from requirements analysis to usability testing of the implemented system [10][22]. Cooper proposed an agile usability approach where the usability engineer first designs the user interface using traditional usability engineering methods before handing off the design to the developers to implement [18]. The problem with this approach is that it can prevent developers from starting work right away and conflicts with the principle of frequent delivery of working software to customers. Agile practitioners such as Patton have introduced approaches where developers are given some training in usability methods so they can both design and implement the system [21]. One potential problem with this approach is that if developers are both designing the UIs and implementing them, they may tend to sacrifice usability for ease of development [6]. In addition, these developers trained in usability may still develop subpar systems as usability engineering is a distinct discipline that requires training and experience to do well.

Others such as Chamberlain, Martin, Miller and Sy have highlighted the importance for having a separate usability engineer working within the team for such UI-intensive systems [5][15][16][17][19]. In this approach, one or more usability engineers work within the team in parallel with development. In this case, one problem is how to balance the different needs of usability engineers and software developers throughout development as issues such as new requirements, tight deadlines and limited resources emerge. Communication between the different team members is critical—something acknowledged in agile methods as noted by the importance placed on *face-to-face communication* [3]. For teams where usability engineers and software developers work in parallel, communication and collaboration is critical for the interface design track and software development track to remain in sync [7][19]. Another focus of this work is developing the proper methods and communication mechanisms for both teams to remain in sync.

One of the key features of agile methods is the importance placed on *working software as the primary measure of progress* [3]. Integrating usability into agile methods can introduce competing goals between development and usability—particularly when there are distinct roles that are working in parallel. This also relates to the agile principle: *simplicity--the art of maximizing the amount of work not done--is essential.* Simplicity in the user interface often does not align with simplicity in the implementation. However, power imbalances can arise in an agile team with usability engineers not having adequate say in how the design and implementation of the system proceeds [5]. Addressing this problem requires committed, motivated team members—also important in agile teams as defined in the agile manifesto. It also requires an understanding of what the different goals of a project are and where usability lies in terms of importance to delivering value to the customer. This also relates back to the issue of communication and collaboration between usability and development.

Despite some of the conflicts described above, both areas are committed to delivering value to the customer where value can be composed of many factors including having highly usable software. This work will look at how they can be addressed in practice.

III. INTEGRATION APPROACH

Based on the analysis of the convergence and divergence points between agile development and usability engineering presented in section 2, we find that an approach that integrates usability engineers into an agile development organization needs to support:

- Proper prioritization of goals to guide design decisions
- Balancing design decisions to reflect high level goals
- Synchronization of usability and development activities
- incremental UI development while maintaining a cohesive overall design

In this work, we have looked at agile methods and usability engineering from a foundational standpoint by comparing the two based on their respective core values and principles (see Figure 1).

Based on this analysis, our past work on agile usability methods, and other research on agile usability methods, we have developed our own instantiation of agile usability—XSBD that draws on existing practices from both areas. We implemented, and continually improved this process by applying it in practice, and worked to meet the four requirements above. Based on the results, we will present several changes, additions and clarifications to the principles in the agile manifesto that summarize core benefits of XSBD. This will lead to further instantiations and refinements of the XSBD process in future efforts.

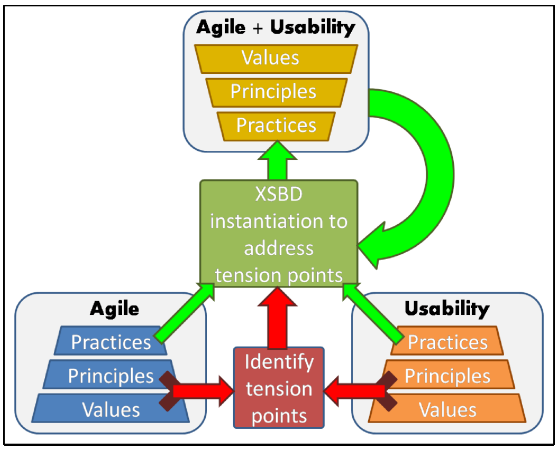

Figure 1. Comparative analysis of core principles leads to XSBD instantiation which contributes to agile usability principle and value development.

IV. XSBD PROCESS DESCRIPTION

This section provides a brief overview of the XSBD process and leads into an extended description of the process in use in section 5. The XSBD process was originally developed by Lee and McCrickard at Virginia Tech [12][13]. It draws on practices from scenario-based design—an established usability engineering methodology and XP and Scrum—two of the most widely practiced agile processes [2][22][23].

In XSBD, the primary addition to the agile team is the usability engineer. The usability engineer works with other members of the team including the developers, QA, and customer representative. The primary responsibility of the usability engineer is to ensure that the user interface of the end system satisfies the needs of the end users.

In the XSBD process there are separate, but synchronized usability and development tracks similar to the parallel development tracks described by Miller [16]. The usability engineer works one iteration ahead of the developers so designs can be delivered for the developers to start implementing at the start of each iteration. Through careful coordination of the development and usability, the XSBD team can optimize its velocity while still developing a system that meets high level design goals—making balanced design decisions that incorporate the oftentimes conflicting needs of the developers and usability engineer.

The key design representation in the XSBD process is the central design record (CDR), which is used to support synchronization activities and help the usability engineer plan and run usability evaluations.

A. User interface design through the CDR

The CDR consists of the set of mockups, scenarios, claims, and design goals. Scenarios, as used in scenario-based design, are narratives describing the activities of one or more persons and include information about user goals, actions and reactions [20][22]. Scenarios provide rich descriptions of how the system will function—useful for when the system is initially being designed—and can show how multiple features tie together to form a complete workflow. Claims are feature descriptions that include design tradeoffs that highlight the potential positive and negative effects of the user interface. Claims are used by the usability engineer to record design rationale for critical parts of the user interface and are later used to plan usability evaluations. One critical component of the CDR developed during this effort is the direct mapping between design goals, claims and usability testing results (See FIGURE 2). This allows usability engineers to focus on and make only the most value-adding usability features. This is critical in an agile development process where the focus is on delivering the most important parts of the software to customers in the shortest time possible.

B. Collaboration through the CDR

A successful development effort hinges on the proper balance between the potentially competing concerns of different stakeholder groups. As a record of design decisions made to the interface, the CDR— which is maintained by the usability engineer and is shared explicitly through written artifacts and indirectly through verbal communications—also acts as a communication point between and among developers, evaluators and clients that helps reduce the chance of miscommunications and misunderstandings [13]. In planning meetings such as those in Scrum, direct links between goals and design decisions can make the impact of usability changes clear in terms of high level goals. Key ways that this happens are detailed in section 5.

C. Usability evaluations through the CDR

Popular agile methods like extreme programming (XP) include the concept of test-driven development and automated testing [2]. Code tests can be aggregated and automatically run whenever the implementation for a particular feature is completed. This allows for very quick verification of new functionality using an increasingly complete test corpus, which helps agile teams develop software more quickly while maintaining functional correctness. Unfortunately, usability testing—which verifies that usability goals are met and help uncover usability problems and new requirements—cannot be similarly automated. These tests are typically run with real people; so in XSBD the usability engineer uses claims to help decide what aspects of the user interface to evaluate and why.

Claims in the CDR are tied directly to high level goals and are used to analyze and reason about specific interface feature tradeoffs. Light-weight usability evaluation methods such as expert walkthroughs and heuristic evaluations test claim tradeoffs and can then be shared with other team members to help in making decisions about what changes should be made given the available resources in a particular iteration. In this way, the usability of the design can be validated at regular intervals to prevent entropy in the overall interaction design as the system is incrementally developed.

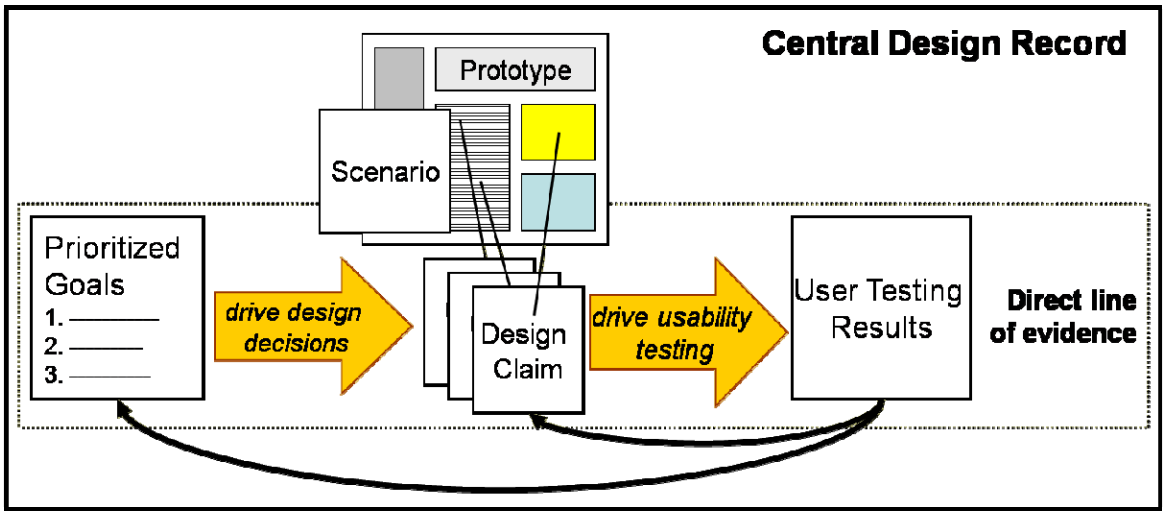

Figure 2. The CDR directly links prioritized goals to design decisions validated by user tests.

V. XSBD IN PRACTICE

This section provides an overview of how the XSBD process (described in section 4) was used and improved by using it in a development project at Meridium.

A. Development Project Overview

The team at Meridium using the XSBD process was tasked by the client with developing a touch screen application to improve a paper-based work order system. This system was meant to improve on an existing system where mechanics would manually enter work order data on a paper form which would then be entered into a company database by clerks. The new system was to have mechanics directly enter work order information into the company database through the touch screen.

Meridium is still in the process of transitioning some of its development teams to agile. As a result, many members of this team were unfamiliar with agile processes. In addition, team members were not used to having a dedicated usability engineer on the team. In the weeks before the project, we held several meetings to introduce the XSBD process although team members had some difficulty understanding the process before actually starting to use it. The team required several weeks to get acclimated to working with each other and with the XSBD process before implemented work began being delivered consistently. The team consisted of two developers, one quality assurance (QA) person, one usability engineer, one project lead, one product lead, and one documentation person. The customer representative worked with us remotely—participating in weekly review/planning meetings and through email. Team members devoted between 15-40 hours a week on the project depending on their other commitments at Meridium.

B. Research methodology

We used an action research methodology to study and refine the XSBD process. Action research is grounded in the concept that complex social processes are best studied by continuously introducing changes and observing effects on the phenomena of study [1]. Action research is divided into a series of five stages forming a complete cycle that is continuously iterated upon. Problems are first identified and *diagnosed*. In the *action planning* stage, researchers and participants identify what actions will mitigate those problems before moving on to the *action taking* stage where those actions are implemented. In the *evaluating* stage, researchers review the results of those actions and how well they addressed the specified problems. In the *specifying learning* stage, researchers develop and build on theories based on the results of this and past cycles.

To identify problems, critical incidents were recorded throughout the development process. Critical incidents are observations of behavior that have significance to the phenomena being studied [8]. These were collected by the researcher through observations during daily meetings, planning meetings and other regular team meetings. We also had project team members record their own critical incidents through an online reporting form [8]. Each critical incident contained the following information: the role of the person writing the incident, the date of the incident, a description of the incident, and an explanation of how this positively or negatively affected the team. We also conducted individual post-project interviews of team members to get their summative experiences of using XSBD.

C. Implementing XSBD

This section describes how the process was used and refined over the course of the development effort. Results are summarized with respect to the four requirements described in section 3.

1) Defining and using shared goals. It was important for team members to develop and be cognizant of the key design goals of the system. This allowed the team to more easily make design decisions when there were competing

concerns. This was important as members of the team had different backgrounds, personal motivations, and mindsets. For example, post-project interviews showed that developers were primarily concerned with ensuring functional correctness and robustness of the system, while the usability engineer's primary concern was in meeting end user needs of the system. Sometimes these conflicted. The goals for the touch screen project were defined as follows:

> **[1] Time Efficiency**
> System interaction must take no more than 2-3 minutes on average.
>
> **[2] User Acceptance of Technology**
> The interface must make sense, be easily learned, and provide low frustration.
>
> **[3] High Quality Data**
> Captured data must be accurate and complete.

The goals are shown from most important to least important. Usability was critical to the success of this particular system so the most important goals were tied to usability. The third goal, which was not directly related to usability, was requested by the customer because he wanted more accurate data to be stored in the company database by having mechanics directly input the work order information. Where possible, metrics were tied to the high level goals to make it more clear to team members that they were met. For example, one metric tied to the first goal was that mechanics needed to take between 2-3 minutes to input a work order because it was important to maximize the amount of time they are working on jobs rather than working on administrative tasks. The QA person noted that:

> "…those goals were always kept in mind, especially in the user trials where we would take the program and let people who weren't involved with the project try it out and see how it works"

Setting the shared design goals allowed team members to have a common understanding of which aspects of the system were most important to the customer. For different types of systems, usability-focused goals may be lower on the priority list—meaning that usability would have less influence over the design. Prioritized goals also helped the team to prioritize which fixes to make when there was a limited amount of time available to complete all required system changes.

Over time, the goals were internalized by most team members and were often used to direct conversations about design decisions throughout the project. However, they were never used as, nor were they meant to be used as unbreakable commandments. Different concerns sometimes overshadowed the goals such as when implementation problems arose. The project manager stated that:

> "…sometimes when you want to get functionality then those goals are not at the top of your priorities. But they were all aware of them and really did strive towards them."

2) Balancing design decisions. Claims proved to be a useful way to capture design decisions and address conflicting opinions on the design. An example of one of the claims captured in the project is listed below:

> **Popup warning**
> + gets user attention
> + lets the user know why they can't move forward
> + avoids frustration of not knowing why program won't let you make a move
> - BUT it can be disorienting
> - BUT it can be too flashy and annoying

The usability engineer always worked to ensure that design decisions reflected the high level goals of the system. Key design decisions were captured in the form of claims. The usability engineer noted that:

> "They helped me keep focus and overall they helped me get the evidence I needed to open up discussions with teammates about the interface."

Claims allowed the usability engineer to provide other team members with rationale for design features and describe why they were important. They provided a platform for discussion about features and their role in the system. This helped in prioritizing the backlog of features that were planned for subsequent iterations.

In addition, since the user interface is the most visible portion of system under development, team members often gave their own opinions about how the interface should be designed based on their own past experiences and knowledge of similar systems. Many times, this sort of feedback was valuable. However, this sometimes resulted in tensions between the usability engineer and developers about how the system should be designed. Claims were used to resolve these kinds of disagreements. If another team member proposed a change to a system feature as designed by the usability engineer, the usability engineer would record the suggestion in the form of a claim. Typically, the original design of the usability engineer would be used initially and would be evaluated. If a problem was uncovered, the proposed change would then be considered.

3) Synchronizing usability and development. Combining the efforts of usability engineers and software developers introduced the problem of design drift—where the implemented system differed from what was initially designed. This problem quickly became apparent in this effort as it relied on tightly interdependent parallel efforts.

This problem was somewhat mitigated by the fact that development in an agile process is broken up into smaller pieces. The project was divided into two releases with the first release focusing on the data entry piece of the system and the second release focusing on the manager approval piece of the system where managers approve of work order information input by the mechanics. The first release covered two scenarios with each scenario taking approximately 2 iterations to complete. The scenarios were written by the usability engineer to describe the key task flows in the system. Stories were written and managed by the developers that covered whatever scenario was being developed in the current iteration.

However, design drift was mainly addressed through synchronization points. Enforced synchronization points through regular review and planning meetings, with all team members including usability engineers, ensured that team members could review and compare designs to implementations, resolving any differences to ensure that they were in sync. These meetings included daily meetings, longer iteration review meetings (during which the work of past weeks could be reviewed), and feature team meetings (during which the upcoming iteration was planned). The regularity of these meetings ensured that differences were found and addressed early on.

Opportunistic synchronizations points between usability engineers and software engineers were also important to resolve issues such as when there are questions with the user interface mockups or when there are differences between the implemented system and the design. These consisted of impromptu face-to-face meetings and electronic communications. The usability engineer stated that:

> "[the developer] would be developing a mockup that I just gave him, I'd be working on a next iteration of the mockup and all of a sudden I'd realize that [he] developed something different than what I prescribed. "

In the end, the synchronizations allowed the parallel efforts to proceed relatively smoothly. In the post-project interview, one of the developers found that:

> "It's great that the developer doesn't have to worry about how the UI is going to look like. When you are developing from scratch and you have time constraints, and you have a design, then you can do it efficiently. "

In addition to having synchronization points, it was important to have the team use a unified work planning system where both developer and usability tasks were estimated and prioritized together. This helped the project manager see how both developers and the usability engineer were progressing and ensure that proper handoffs of designs and implementations between the two were occurring.

4) Incremental user interface development. Maintaining direct links between goals, design decisions, and usability evaluations through the CDR, allowed the usability engineer to design the user interface incrementally while still maintaining a cohesive overall design (see Figure 3). The usability engineer focused on testing only the design decisions linked with the high level goals of the system. In this case, a design for a popup selection box on the touch screen is shown to not slow down the user during the main work flow. This targeted testing helped in maintaining project velocity as non-essential user interface changes could be deferred or dismissed. It also made it easy for developers to understand the rationale behind design decisions by seeing the link between goals, design changes and usability testing results. Several team members began the project with little understanding of what the usability engineer did and why. However, by the end of the project one developer stated that:

> "If something was to be changed, we were like: 'why do we want to change this?'. And if the rationale and user reactions could be laid out, everyone felt like that was the right thing to do."

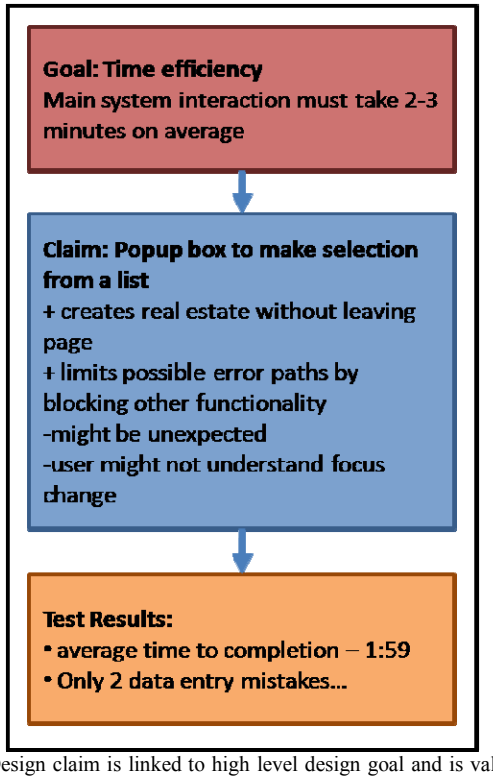

Figure 3. Design claim is linked to high level design goal and is validated through usability testing.

Overall, agile team members found that incrementally developing the user interface was valuable in that it allowed problems to be found early on, uncovered new requirements and helped solidify ill-defined ones. The usability engineer

also found that developing incrementally while maintaining the CDR allowed her to:

> "…get a decently consistent interface that fits the customer better."

The usability engineer believed that she would have gotten a more consistent and cohesive design if she had been able to do a more complete requirements analysis and design phase at the beginning of the project, but thought that the end result fit the customer's needs better. In addition, developing in this way also allowed the usability engineer work within the agile team effectively.

Usability testing was manually run by the usability engineer as usability tests typically cannot be automated and managed like they are in agile processes like XP [2]. The usability engineer used claims to help decide what aspects of the user interface to evaluate and why. Claims were only written against features that were directly linked to high level design goals, and only those claims were evaluated at the end of an iteration. Limiting evaluations in this way using claims allowed the usability engineer to more efficiently run tests that maximally contributed to the overall usability of the system. The types of evaluations run varied for project, but relied primarily on light-weight evaluations with 3-5 participants selected at Meridium for each evaluation.

VI. Discussion and Conclusions

In this effort, we developed and improved an approach to integrating usability into an agile team—XSBD. We first looked at where some of the key convergence and divergence points were by analyzing agile and usability approaches from a foundational standpoint. We then applied it in a development effort at Meridium and actively observed and evaluated how those points were addressed using an action research methodology. Based on its use at Meridium, we present the following conclusions:

- Maintaining and collectively agreeing on a prioritized list of design goals will help to resolve conflicts between usability engineers and software developers by making clear where customer priorities lie.
- Claims are an effective and lightweight mechanism for capturing design rationale and storing alternate design proposals for specific features.
- A combination of enforced and opportunistic synchronization points actively maintained by agile team members is important in mitigating design drift between parallel development and usability tracks.
- The central design record, by making explicit the link between design goals, design decisions and usability testing results, can help usability engineers develop a more cohesive user interface while designing it incrementally.

Note that the XSBD process is meant to be used for projects with a non-trivial UI component where usability is one of the key quality concerns. Its successful application in this effort is evidence of that.

Looking broadly from a principles perspective, we also present the following list of additions and changes to the principles from the agile manifesto for any agile project that integrates a usability engineer into their team [3].

- *Balancing the often competing needs of development and usability personnel based on project goals is critical.*
- *Deliver UI designs frequently, to get continual feedback from end users and customers to improve the product.*
- *Business people, software developers and usability engineers must work together daily throughout the project.*
- *The most efficient and effective method of determining the usability of the system is through usability testing—preferably involving end users.*
- *Working software that meets high level goals (functional, usability, performance) is the primary measure of progress*
- *Continuous attention to good design, both in terms of the code and the user interface, enhances agility.*
- *Appropriately balancing simplicity in the code and the UI is essential.*

VII. Future Work

We are currently planning to deploy the XSBD process in all of the agile, UI-intensive development efforts at Meridium while continuing to research ways to improve the process and better understand the challenges involved in integrating usability and agility. We will focus on improving the CDR to better support cohesive interface design and work to further improve synchronization between developers and usability engineers. In addition, we are working to expand our focus beyond just integrating usability engineers and agile developers by looking at how other areas in an enterprise-level organization like Meridium work with and within agile usability teams. These include areas such as quality assurance, documentation and product management.

We are also working to apply the XSBD process to a broader spectrum of projects in different application areas and with agile teams with varying levels of knowledge of agile and usability methods. This will help verify the broader applicability of the XSBD process and the findings presented in this work.

VIII. ACKNOWLEDGEMENTS

Thanks to the development team for their valuable insights during the project. This material is based upon work supported by a National Science Foundation Small Business Technology Transfer Phase I Grant No: #0740827.

REFERENCES

[1] Baskerville, R. L. "Investigating information systems with action research". Communications of the AIS vol. 2(3es), 1999, 1-32.

[2] Beck, K., and Andrews, C. Extreme Programming Explained: Embrace Change, 2nd Edition. Addison Wesley Professional, Boston, MA, 2004.

[3] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. "The agile manifesto". 2001. http://agilemanifesto.org

[4] Beyer, H., Holtzblatt, K., and Baker, L., "An agile customer-centered method: rapid contextual design". Proc. XP/Agile Universe '04, Springer Berlin/Heidelberg, 2004, 50-59.

[5] Chamberlain, S., Sharp, H., and Maiden, N., "Towards a framework for integrating agile development and user-centred design". Proc. 7th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP '06), Springer Berlin / Heidelberg, 2006, 50-59.

[6] Cooper, A., and Reimann, R., About Face 2.0: The Essentials of Interaction Design, Wiley Publishing Inc., Indianapolis, IN, 2003.

[7] Ferreira, J., Noble, J., and Biddle, R. "Agile development iterations and UI design." Proc. AGILE 2007 Conference (Agile '07), IEEE Press, 2007, 50-58.

[8] Flanagan, J.C. "The critical incident technique". Psychological bulletin, vol. 51(4), 1954, 327-358.

[9] Hartson, H. R., Castillo J. C., Kelso, J., and Neale, W. C. "Remote evaluation: the network as an extension of the usability laboratory". Proc. Conference on Human Factors in Computing Systems (CHI '96), ACM Press, 1996, 228-235.

[10] Hix, D., and Hartson, H. R., Developing user Interfaces: Ensuring Usability through Product and Process, John Wiley & Sons, Inc., New York, NY, 1993.

[11] Jokela, T., and Abrahamsson, P. "Usability assessment of an extreme programming project: close co-operation with the customer does not equal good usability." Proc. 5th International Conference on Product Focused Software Process Improvement (PROFES '04), Springer Berlin / Heidelberg, 2004, 393-407.

[12] Lee, J. C. "Embracing agile development of usable software systems". Doctoral consortium paper in Proc. Conference on Human Factors in Computing Systems (CHI '06), ACM Press, 2006, 1767-1770.

[13] Lee, J. C. and McCrickard, D. S. "Towards extreme(ly) usable software: exploring tensions between usability and agile software development." Proc. AGILE 2007 Conference (Agile '07), IEEE Press, 2007, 59-71.

[14] Martin, A., Biddle, R., and Noble, J. "The XP customer role in practice: three studies". Proc. AGILE 2004 Conference (Agile '04), IEEE Press, 2004, 42-54.

[15] Martin, A., Noble, J., and Biddle, R. "Programmers are from Mars, customers are from Venus: a practical guide for customers on XP projects. Proc. 2006 conference on pattern languages of programs (Plop '06), ACM Press, 2006.

[16] Miller, L. "Case study of customer input for a successful product". Proc. AGILE 2005 Conference (Agile '05), IEEE Press, 2005, 225-234.

[17] Miller, L, and Sy, D. "Agile user experience SIG". Proc. Extended abstracts of Conference on human factors in computing systems (CHI '09), ACM Press, 2009, 2751, 2754.

[18] Nelson, E., "Extreme programming vs. interaction design". Fawcette Technical Publications, 2002. http://web.archive.org/web/20060613184919/www.fawcette.com/interviews/beck_cooper/

[19] Nodder, C. and Nielsen, J. "Agile usability: best practices for user experience on agile development projects". Nielsen Norman Group, Fremont, CA, 2008.

[20] Obendorf, H. and Finck, M. "Scenario-based usability engineering techniques in agile development processes". Case study in Proc. Conference on Human Factors in Computing Systems (CHI '08), ACM Press, 2008, 2159-2166.

[21] Patton, J. "Hitting the target: adding interaction design to agile software development". Proc. Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '02), ACM Press, 2002, 1-ff.

[22] Rosson, M. B. and Carroll, J. M. Usability Engineering: Scenario-Based Development of Human-Computer Interaction. Morgan Kaufman, New York, NY, 2002.

[23] Schwaber, K., and Beedle, M. Agile Software Development with SCRUM. Prentice Hall PTR, Upper Saddle River, NJ, 2001.