Embracing Agile Development of Usable Software Systems

Jason Chong Lee and D. Scott McCrickard
Center for Human-Computer Interaction and Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106 USA
{chonglee, mccricks}@cs.vt.edu
1-540-231-7409

ABSTRACT
Agile software development has largely ignored or been unable to address usability. This work presents a development process based on extreme programming—from agile software development, and scenario-based design—from usability engineering. This approach will allow developers in both fields to communicate and work together to efficiently design usable systems.

KEYWORDS
extreme programming, scenario-based design, agile development, usability, central design record

1 INTRODUCTION
The growing importance of computing systems in our daily lives is reflected by the reality of the multi-disciplinary nature of system design [18, 19]. One critical example of the problems that can arise from this is the disconnect between software engineers and usability engineers. Software engineers focus on the design, implementation and maintenance of software systems but often marginalize the design of the human-computer interfaces through which those systems are used. Conversely, usability engineers focus on developing systems so end-users can use them effectively but largely do not account for the underlying system design, implementation or market-driven forces that guide much of software engineering.

This research will address the problems associated with multi-disciplinary system design—focusing specifically on software and usability engineering—by developing a design process that supports the effective creation of usable software-based systems through common practices and toolsets derived from both areas. This work draws from extreme programming (XP), an agile software development process, and claims-centric scenario-based design (SBD), a usability engineering process. Although the two processes emerged from different areas of study, they share similar motivations and the combination of the two can result in a process that supports team collaboration in effectively developing usable computer systems.

This paper will first summarize software development processes and the emergence of agile software development methods. It will then give a brief overview of usability and motivate the need for agile methods to account for interface usability. The paper will then present an overview of the completed and proposed work.

2 BACKGROUND & RELATED WORK
Both software and usability engineering are relatively new fields that arose in response to the increasing complexity of developing computer systems. This section presents some of the foundational ideas of software engineering, the factors that resulted in the development of agile software development methods and the increasing importance of usability in such systems. This background information will motivate the need for and importance of common tools and practices to support both areas when developing systems.

2.1 Software Engineering and Agility
In software engineering, prescriptive process models are used to control the chaos that can surround software development. Software processes and methodologies for large software systems adopt a ‘divide and conquer’ approach to development with different teams working on different phases of development such as requirements analysis, implementation and testing. For example, the waterfall model of development, one of the oldest software development paradigms, defined the phases that are central to most modern software development processes. Boehm [4] proposed the spiral model of development, an evolutionary risk-driven process that incorporates some of the systematic aspects of waterfall development. The Unified Modeling Language (UML), developed by Rumbaugh, Booch and Jacobson, is a notation used in modeling and developing object-oriented systems and has become an industry standard [21]. The Unified Process, derived from UML, is a framework for iterative, incremental object-oriented software development and is becoming increasingly popular within the software development community.
Many existing processes are based on the premise of the increasing cost of change curve which states that system changes will be more costly the later in the development cycle a system is in [3, 11]. This results in a focus on upfront requirements analysis and design. However, these processes are unable to account for continuous requirements and system changes that are often needed throughout the development process. This has resulted in the emergence of agile software methodologies. Agile methodologies focus heavily on quick delivery of working software, incremental releases, team communication, collaboration and the ability to respond to change [2]. One stated benefit of agile methods is a flattening of the cost of change curve throughout the development process. In fact, the Unified Process was partly developed as a way to integrate the best practices of traditional software development and agile methodologies [21].

A number of different agile methodologies have been developed around these principles. Adaptive software development is based on complex adaptive systems theory and focuses on collaboration and team self-organization [13]. SCRUM focuses more on the project management side of agile development processes. One of the most widely adopted agile processes is Extreme Programming, which treats design as a transient artifact that can and should be continually modified [3]. Extreme Programming eschews upfront design processes such as those in traditional software engineering processes and instead proposes an evolutionary design process with tightly integrated designer/customer teams that encourages system changes at all phases of the development process. It rests on the values of communication among project stakeholders, simplicity in the design, continuous feedback through test-driven development and through evaluations by customers and other team members, courage through minimal design and a willingness to address changes, and respect for other team members and stakeholders.

2.2 The Need for Usability

One shortcoming of many software development processes is their marginalization of usability issues. This is especially true of agile software development methodologies which have only emerged in the last decade. Usability engineering is concerned with developing interfaces that people can use efficiently and effectively. It deals with issues such as system learnability, efficiency, memorability, errors and user satisfaction [16]. Numerous usability engineering methodologies share many similar characteristics such as a focus on end users and iterative design coupled with analytic and empirical evaluations [9, 12, 22].

One such approach is Rosson and Carroll’s Scenario-Based Design (SBD), a design process in which scenarios, narratives describing a particular task, are used in conjunction with design knowledge components called claims, which encapsulate the positive and negative effects of specific design features as a basis for creating interactive systems [6, 22]. Unlike other knowledge capture mechanisms like patterns and use cases [5, 7], claims provide compact, designer-digestible packets of knowledge ideal for use in time-critical design activities [24]. They also have the benefit of making trade-offs in design features explicit.

2.3 Agile Software and Usability

Many of the usability methodologies do not adequately address the problems encountered in practical software development environments. Given the importance of software engineering and usability in system development, many researchers are looking at ways for developers in both fields to cooperate with one another. Seffah and Andreescu propose an education focused approach where software engineers use a cost-effective educational framework to learn user-centered design concepts [23]. Faulkner and Culwin propose the adoption of human-computer interaction principles as an underlying principle to guide the development computer systems [10]. However, these approaches draw mainly on concepts from either software engineering or on usability engineering, which discourages adoption and cooperation between the two disciplines. Bass and John propose the use of a specific set of usability scenarios and related patterns to act as a link between the software and usability domains [1]. However, this approach was not developed to function within an agile development framework.

The problem of addressing usability issues within agile software development is also beginning to be acknowledged and addressed [17]. In fact, development processes from both areas such as extreme programming and scenario-based design share many of the same foundational concepts including iterative development and a focus on users. However, a combined approach is proving to be difficult because agile methods, which are incremental and iterative in nature, do not support any kind of comprehensive overview of the entire architecture which is an important part of making consistent and usable interfaces [8]. To address architectural issues, many usability methods advocate in-depth requirements and task analysis processes at the beginning of the design process. In fact, Constantine advocates a combined usability and agile software development process that begins with interface design and then continues with existing agile software development processes. The problem with this approach is that the interface usability design process becomes a bottleneck in the overall design process and violates many of the accepted tenets of the agile development philosophy.

3 ENABLING AGILE USABILITY

The challenge of combining usability engineering and agile software development will be in finding ways to support a process that supports the definition of a user interface architecture without sacrificing the efficiency and flexibility of agile development methods. Our approach leverages the
flattened cost of change curve that agile software methods support and treat usability and software engineering as concurrent and coordinated processes in a single development framework. We draw primarily on Extreme Programming, an agile software development process, and claims-centric scenario-based design, a usability engineering process. In developing this process, we address two primary issues:

1. How can interaction architecture issues be addressed in an agile development framework?
2. How can usability evaluations be conducted in an agile framework without bottlenecks the development process?

The first question has been addressed by combining a design representation based process based on scenario-based design from usability engineering to the extreme programming framework. The proposed work will address the second question.

Figure 1. Example claim from assistive technology project accessed from LINK-UP.

3.1 Scenario-Based Design to the Extreme
Interaction architecture level issues are addressed by incorporating past work on design representations in scenario-based design into the extreme programming framework. Scenario-based design is an iterative, user-centered process—qualities that are shared by many agile methodologies including XP. Our adaptation of scenario-based design is supported by LINK-UP, a web-based integrated design environment and knowledge management system (See Figure 1) [15, 20]. A core feature of LINK-UP is the central design record (CDR), a design representation that makes explicit where and how handoffs occur in the development process and highlights design decisions that need reconsideration during subsequent development iterations. The CDR consists of an organized set of scenarios describing different usage situations, claims which highlight positive and negative effects of design features, and overall design goals (See Figure 2). In several studies, the CDR was found to help usability designers to iteratively improve designs through targeted feedback from evaluators and support communication between different stakeholder groups by making design intentions explicit [14].

Pilot Case Study Results. The purpose of the pilot study was to evaluate how effectively the CDR allowed developers to address interaction architecture issues in an agile development environment. Seven undergraduate students worked in teams of two and three to develop location-aware notification systems over a period of four two week iterations. Location-aware notification systems are secondary systems that provide users with valued information and services based on their current location. These systems were developed for clients at Virginia Tech in a realistic development environment. They included a system for locating resources in a library, a system to allow alumni to tour the campus and a system to allow people with limited mobility to navigate in and around buildings. Students iteratively developed CDR representations of their designs within the existing extreme programming framework to address usability issues. Extreme programming includes accelerated design sessions known as ‘planning games’ to determine key features to support and to identify and assign development tasks [3]. An additional planning game called the ‘architecture planning game’ was incorporated into the development process during when the student developers created and modified their CDRs—which included key interaction scenarios and claims for the interfaces. The CDR was initially useful to the student developers in defining the task flow and specific interface features. Both new claims and existing claims from LINK-UP were used to highlight specific design tradeoffs. However in subsequent iterations, developers did not use the CDR and instead focused their efforts on the functioning system prototype. Although this is in-line with the XP concept of design representations as transient artifacts, much of the usability benefits of the CDR and LINK-UP were lost. This led to some major usability issues in the final prototypes that were never addressed.

3.2 Proposed Work: Agile Usability Evaluation
In the pilot study, the utility of the CDR diminished because of the lack of a clear connection between interface features described and usability evaluations that were conducted on it. A key addition to the process will be to define how this connection will be made to support iterative enhancement to the interaction architecture based on usability evaluations. By extending the concept of test-driven development to user interface testing and integrating it with the CDR design representation, developers will be able to validate system usability and iteratively enhance designs without requiring extensive up-front requirements analysis or interaction architecture design. Rather,
requirements analysis will proceed hand-in-hand with interface design as requirements emerge and evolve.

One of the key concepts of extreme programming is test driven development [3]. When implementing features, developers first write test cases to validate the functionality of new code classes and methods before beginning any actual implementation. This growing suite of tests is used to verify code functionality. Only code that passes the entire test suite can be added to production code. This allows developers to make code modifications based on changing requirements and ensure that all other parts of the code still function correctly. This process gives developers continuous feedback and allows them to reliably make changes to code at all stages of development. A similar process could work with respect to usability evaluations while still supporting overall architecture views and consistency through the CDR. The organization of the CDR naturally supports both high level system views (through core claims and scenarios) and feature-level views (through information and interaction claims). Developers can leverage this view to develop usability tests to validate specific claims in the CDR. As developers associate usability tests with the claims in the CDR, they will build up a test suite analogous to the one used to validate program code. Both analytic and empirical evaluations from the evolving test suite would validate the features defined by each claim or identify areas for improvement in the interface that have to be made.

One important difference between the usability test suite and the code test suite is that the usability test suite cannot be completely automated since many evaluation techniques require participant feedback and data. Although an additional module will be developed for LINK-UP to manage and organize usability testing that needs to be run on the interface, as the design progresses, the number of claims and associated usability tests that need to be run would become large and difficult to manage. One way to mitigate this problem will be to use the relationships between the claims to identify which areas of the interface need to be evaluated in the current evaluation. Looking at Figure 2, if the ‘Colored-news alert’ claim is added in the current iteration, then it is likely that only claims that are directly related, namely the ‘text-based links’ claim and the ‘keyword highlighting claim’ would have to be evaluated for usability assuming that other aspects of the interface have already been evaluated in previous iterations. Wahid et al have defined specific relationship-types that exist between claims which will be used to determine what aspects of the interface have to be evaluated, or reevaluated at each iteration [25]. A second way to mitigate this problem will be to leverage low-overhead, discount usability methods to validate claims. These analytic methods would be easier to run within short time-frames while providing useful usability feedback. Empirical evaluations, which are often more time and resource intensive, could then be spaced out between several iterations as necessary.

4 KEY CONTRIBUTIONS

Integrating usability evaluations into the agile development process by leveraging the CDR will allow developers to validate the quality of the system interface in an efficient manner using a design representation-based development process. The proposed solution will make the following contributions:

- Allow developers who use agile software development processes to efficiently address usability issues
- Support collaboration between software engineers and usability specialists by facilitating communication of design intent and design rationale
- Support efficient design representation-based development by leveraging techniques from agile software development

This development process will allow agile software developers and usability specialists who leverage scenario-based design to work together more effectively. It will serve as an important step in finding ways for designers in different disciplines to work together to develop computer-based systems.

REFERENCES


