Integrating scenario-based usability engineering and agile software development

By
Jason Chong Lee

Proposed research in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Applications
at
Virginia Polytechnic Institute and State University

November 2007

D. Scott McCrickard, Chair
James D. Arthur
Robert L. Biddle
Shawn A. Bohner
Manuel A. Pérez-Quiñones
Abstract

In recent years, agile software development methodologies such as extreme programming and SCRUM have emerged as one way to mitigate major risks of software development such as changing requirements, communication breakdowns and cost and scheduling overruns. These methodologies have become increasingly popular within the software development community and have been used in a variety of projects because they have helped in delivering software on-time that meets customer requirements. However, agile development methodologies—which focus on the design, implementation and maintenance of software, have marginalized the role of usability—which focuses on optimizing the interaction and information design of a system. This often results in software systems that meets functional requirements, but are difficult or frustrating to use. There is a need to develop a way to integrate usability into agile development practices without compromising the benefits they bring to software engineering.

I have developed an approach to integrate extreme programming (XP)—a widely practiced agile design method with scenario-based design (SBD)—a well established usability engineering process. Key to this extreme scenario-based design (XSBD) approach is the central design record (CDR), a shared design representation I developed based on my work in rationale-based usability engineering. I hypothesize that this approach will enable the creation of agile development teams that include usability engineers who can work effectively with other stakeholders to develop usable software systems efficiently. This work will address three key issues:

1. How to satisfy the requirements from different stakeholders (specifically usability and software engineers) when they have different motivations, needs and concerns

2. How to maintain project velocity and a focus on the delivery of working software with the addition of usability practices and tools to the development process

3. How to develop & maintain a coherent interface and interaction architecture design in an agile development environment

The successful integration of SBD and XP depends not only on shared design representations, but also on how they are used throughout the development cycle. Proposed efforts will focus on how the CDR can be used effectively in practice and will focus on how usability artifacts and results should be shared with agile developers and how project velocity can be maintained. This work will be validated with a comprehensive case study conducted at Meridium, Inc and results will be disseminated through journal and conference paper submissions and workshops at leading agile and usability venues. Design principles and practices that are developed will be broadly applicable to the agile community and will also contribute to and make connections with the HCI research community.
Contents

ABSTRACT ...................................................................................................................................................... I

LIST OF FIGURES AND TABLES ................................................................................................................... V

1 INTRODUCTION ............................................................................................................................................. 1

1.1 KEY RESEARCH ISSUES ......................................................................................................................... 2
1.2 APPROACH .............................................................................................................................................. 3
1.3 ANTICIPATED IMPACT ............................................................................................................................ 4
1.4 PROPOSAL OVERVIEW ........................................................................................................................... 4

2 RELATED WORK ......................................................................................................................................... 6

2.1 USABILITY ENGINEERING ...................................................................................................................... 6
2.1.1 High-level design goals .............................................................................................................. 7
2.1.2 Scenario-based design ................................................................................................................. 8
2.2 EMERGENCE OF AGILE METHODS ................................................................................................... 10
2.2.1 The Agile Manifesto .............................................................................................................. 10
2.2.2 Agile landscape ....................................................................................................................... 11
2.2.3 Extreme programming ............................................................................................................. 12
2.3 ENABLING AGILE USABILITY ........................................................................................................... 14
2.4 CHAPTER SUMMARY ........................................................................................................................... 15

3 BACKGROUND WORK .............................................................................................................................. 17

3.1 LEVERAGING A DESIGN REPRESENTATION IN LINK-UP ................................................................... 17
3.1.1 Introduction ........................................................................................................................................ 17
3.1.2 Background ....................................................................................................................................... 18
3.1.3 Analytic module development ....................................................................................................... 20
3.1.4 Pilot testing evaluation methods ................................................................................................... 22
3.1.5 Summary of study findings .......................................................................................................... 23
3.2 BRINGING DESIGN REPRESENTATIONS TO THE FOREFRONT ........................................................... 24
3.2.1 Introduction ...................................................................................................................................... 24
3.2.2 Background .................................................................................................................................... 25
3.2.3 Enhancing LINK-UP .................................................................................................................... 26
3.2.4 User study description ............................................................................................................... 26
3.2.5 Summary of study findings .......................................................................................................... 27
3.3 OTHER PRELIMINARY WORK .............................................................................................................. 28
3.4 CHAPTER SUMMARY ........................................................................................................................... 29

4 COMPLETED WORK: AGILE USABILITY WITH THE CDR ................................................................. 30
List of figures and tables

Figure 1. Scenario and associated claim for the notification collage. The claim highlights the tradeoffs of the bulletin board metaphor used in the system. ................................................................. 9

Figure 2. Left: early storyboard of the analytic module; Right: Screen from the functional prototype used in pilot testing........................................................................................................ 20

Figure 3. Basic sequence of tasks of the analytic module of LINK-UP............................................. 21

Figure 4. Norman’s conceptualization of the system image. The designer captures system intentions in a prototype, along with supporting documentation, enabling the study of user interaction with the system................................................................................................................................. 25

Figure 5. LINK-UP's iterative design process and CDR development. Designers start at the center identifying requirements and a target IRC goal. Design iterations through the CDR include design claims which are tested through evaluations leading to convergence of the design and user models. .............................................................................................................................................. 33

Figure 6. Part of the requirements analysis module. The progress/navigation bar is always visible on the right side of the screen to show designers where they are in the process......................... 34

Figure 7. Part of the central design record module. Users create a set of scenarios and corresponding claims for each supported activity. Areas of the CDR module can be accessed in any order...... 35

Table 1. Summary of strengths and weaknesses of LINK-UP derived from three development case studies.................................................................................................................... 37

Figure 8. Key steps in XSBD process. Design artifacts are shown in white..................................... 40

Figure 9. Example claims map. The root concept claim (green) is linked to activity claims (blue) which are linked to implementation claims (yellow). Note the progressive growth through the iterations. (Note that only claim features are included in the diagrams)................................. 43

Table 2. Table shows how the XSBD approach addresses tensions between agile methods and usability. ................................................................................................................................. 46

Figure 10. Usability engineers present to other stakeholders only the parts of the CDR that help them to easily understand and reason about the design................................................................. 51

Figure 11. Cards can be used to present usability problems to other project stakeholders. They can then be used to collaboratively work out what issues can and should be resolved. ..................... 53

Figure 12. Only relevant usability information is presented to different stakeholders to support project management, development, and acceptance testing................................................................. 56

Figure 13. Timeline of milestones and proposed work. ..................................................................... 64
1 Introduction

Software engineering is a relatively new area compared to other engineering disciplines. This fact, combined with the increasingly ubiquity and complexity of software-based systems and how they are integrated into modern society have greatly increased the difficulty of engineering and delivering software that is on-time, on-budget and satisfies the requirements of its users [4][11]. In addition to the technical and business challenges of developing software, the increasing complexity and importance of how users interact with these systems has necessitated the development of usability engineering—which focuses on the interaction design and interface between software systems and users [1][7][25][59][64]. In recent years, agile software development methodologies, such as extreme programming (XP) and SCRUM, have emerged as one way to mitigate some of the risks of software development such as changing requirements, communication breakdowns and scheduling overruns [44]. These methodologies value individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan [7]. These methodologies have become increasingly popular within the software development community and have been used in a variety of software projects because they have helped in delivering software on-time that meets customer requirements. However, agile development methodologies—which focus on the design, implementation and maintenance of the software, marginalized the role of usability—which focuses on optimizing the interaction design, information presentation and other user interface issues. This often results in software systems that meets functional requirements, but are difficult or frustrating to use [65]. Recently, agile software developers have realized the need to integrate usability into their design practices [20][24][52][65][74]. This work will focus on building on and complementing these efforts by exploring ways to integrate usability practices into the agile framework without compromising its benefits to software development.

This work will specifically explore ways to integrate extreme programming—one of the most widely accepted agile methodologies, with scenario-based design—an established usability engineering process. In some ways, these two methodologies share the same foundations in that they both focus on cyclic development processes with close collaborations between different stakeholders. However, the concerns of the agile software developers focus on meeting the needs of customers and are primarily development centric whereas the concerns of usability engineers are focused on meeting the needs of end-users and are primarily interaction-centric. The divergent needs and concerns of different stakeholder groups illustrate some of the inherent problems of multidisciplinary system design. Furthermore, extreme programming and scenario-based design differ in how they approach
interface design. Extreme programming argues against extensive up front requirements analysis because user needs and requirements often change throughout the development process. Developing an understanding of the users and their tasks is seen as an emergent property of the extreme programming process. On the other hand, scenario-based design, and similar usability methodologies require a comprehensive requirements analysis process to gather information about users, the tasks that will be supported and the context in which the system will be used. Successfully integrating the two approaches requires that this problem be addressed.

The goal of this work is to combine extreme programming and scenario-based design into a single development framework that effectively provides the benefits of both approaches at minimal cost. This extreme scenario-based design (XSBD) approach, centers on a shared design representation known as the central design record (CDR). This design representation will allow agile developers and usability professionals to better communicate design intentions and make design tradeoffs when meeting with each other. Usability engineers will work to maintain the CDR and collaborate with software developers throughout the design process to maintain a consistent interaction architecture within the incremental agile framework. This work is summarized in the following thesis statement:

**Integrating agile development processes and usability engineering processes through shared design representations, continuous interaction architecture improvement and targeted, light-weight usability evaluations will enable agile teams to leverage usability design practices without violating the core agile design principles.**

1.1 Key research issues

The successful integration of extreme programming and scenario-based design depends on a number of different factors. The agile manifesto, the core principles of agility developed by a group of leading agile practitioners, stresses the importance of individuals and interactions and collaboration among project stakeholders [6]. This focus on communication is also a key part of developing usable software. One focus of this work will be to determine how to enable agile developers and usability engineers to communicate with each other better so they can make informed design tradeoffs that take into account their differing viewpoints, motivations and concerns. The agile manifesto also stresses the importance of continuously delivering working software. This work will also address the problem of how to continuously deliver working software with the addition of usability engineering practices and tools. Finally, this work will explore how to develop usable software while still meeting the functional and business needs of the customers. These three issues, that if addressed will validate the above thesis statement, are summarized below:
1.2 Approach

To address the questions above, my general approach will draw on the design artifacts and best practices of scenario-based design and extreme programming. This approach has been divided into two phases. The first phase was to identify key design artifacts of both SBD and XP, exploring how they can be interrelated through a shared design representation. It focused on how this representation could be used to support usability engineering in an incremental development cycle. This work has been completed and is discussed in Chapter 3 and 4. The second phase of this work, which describes the proposed work in this document, will be to determine how the shared design representation can best be integrated into a practical agile development environment to support communication between usability engineers and other stakeholders without affecting project velocity.

The approach depends on a shared design representation known as the Central Design Record (CDR)—originally developed as part of my work in rationale-based usability engineering [38][39][40][41][42]. This design representation draws on artifacts from both SBD and XP. It provides multiple perspectives on a design which are then used by stakeholder groups to communicate design intentions and make design tradeoffs. Design goals expressed in terms of critical parameters [58], and a root concept document provide the highest level view of the design. Scenarios provide the next level of detail by describing usage situations in narrative form [71]. These scenarios are further broken down into features which are prioritized and implemented within the incremental agile development cycle. Features can be expanded into claims, which describe specific design tradeoffs and can be used to analyze project risks. This design representation is described in Chapter 4.

The successful integration of SBD and XP depends not only on shared design representations, but also on how it is used throughout the development cycle. From an agile software development standpoint, this work will explore how SBD can fit within an agile framework with respect to development/planning meetings and through short, incremental development cycles. From a usability standpoint, this work will explore how a consistent and coherent interaction architecture can be
maintained in an agile framework by developing a CDR that is validated through lightweight usability evaluations to form a feedback cycle that can continuously and consistently improve the interaction design.

### 1.3 Anticipated impact

The overall goal of this project is to provide a framework within which agile software developers and usability engineers can work together to efficiently create usable software systems through common practices and toolsets from both disciplines. This will be achieved through the use of well defined scenario-based usability design practices that are integrated within an agile framework derived from extreme programming and shared design representations among different stakeholder groups. Design principles and practices will be developed that can be applied to agile usability in general, and will build on and reinforce the growing body of work in this area [17][20][24][28][52]. Agile practitioners who use this approach can expect the following benefits:

1. Higher client/end user satisfaction because of a development process that focuses on continuous delivery of value-adding software, is flexible and responds to changing needs—resulting in products that meet users’ functional and usability needs.

2. Improved communication and collaboration between stakeholders including software engineers, usability specialists, business analysts and customers through shared design representations and tools that prevent wasted development effort and leads to fewer customer problems with the end product.

3. Continuous, measurable improvement to product usability through design targets and evaluations that is comparable to existing agile development metrics

4. A living record of rationale and design decisions made throughout the design process that can be shared and reused, thereby saving time and effort in later development projects.

### 1.4 Proposal overview

**Chapter 2 – Related Work** – This chapter summarizes previous work that provides the groundwork on which this work is based. It covers work related to usability engineering and agile methodologies and provides an overview of other research to integrate agile software development and usability.

**Chapter 3 – Background Work** – This chapter summarizes my initial work in scenario-based development and knowledge reuse. It describes my work on the LINK-UP system, an online scenario-
based toolset that supports the SBD process. This work led to my initial work on design representations—based on Norman’s work on the *system image* [63] and Carroll and Rosson’s work in scenario-based usability engineering [12][71]. This work and associated research studies, involving students working on semester-long development projects, provided the foundation from which my current research on integrating usability and software development methodologies is grounded [39][40][49][56].

**Chapter 4 – Completed Work: Agile Usability with the CDR** – This chapter summarizes how the shared design representation known as the central design record was developed and how it was integrated into the XSBD process. The CDR was initially added to the LINK-UP system as the main design representation of the scenario-based development process [42]. It describes how the CDR was used by student developers in designing systems and the problems and issues that became apparent. It then highlights how the CDR was modified to act as a central design representation that would be shared by different stakeholders in an agile software development project—with a focus on usability engineers and software developers [38][41].

**Chapter 5 – Proposed Work: Bringing the XSBD Process to the Real World** – This chapter describes proposed additions and improvements to the XSBD development process and how the CDR is used by the developers. This chapter has two key foci. First, it describes how the CDR will better act as a driver for maintaining common ground among different types of stakeholders and allows developers to analyze the system design from different perspectives. Second, it focuses on how agile teams can maintain project velocity while integrating usability practices in their development process.

**Chapter 6 – Proposed Research Case Study** – This chapter describes the research case study that will address the second phase of this work—how agile practitioners and usability specialists can work together within an agile framework to develop usable software systems using the CDR. This study, supported by an NSF Small Business Technology Transfer (STTR) Phase I grant, will involve a joint development effort with Meridium, Inc. [51][55].

**Chapter 7 – Summary** – This chapter summarizes key contributions and broader impacts of this work. This includes a timeline of the proposed work and a description of key deliverables.
2 Related work

This chapter reviews literature relevant to this proposed work. It first introduces key concepts of usability engineering—detailing the scenario-based design approach and its core theoretical and practical concepts. It then gives an overview of agile methodologies, their guiding principles and describes key concepts from extreme programming. It highlights the need for developing ways to integrate usability into agile methodologies and gives an overview of ongoing work in this area within the agile community.

2.1 Usability engineering

Usability is closely related to Human-Computer Interaction (HCI)—the area of study that focuses on understanding how people use computer devices, and how such devices can be made useful and usable [12]. It bridges the social and behavioral sciences with the engineering of computer-based systems and is thus an inherently multidisciplinary endeavor. Usability engineering is specifically concerned with the practical application of knowledge about how people use systems to develop interfaces that can be used efficiently and effectively. It deals with issues such as system learnability, efficiency, memorability, errors and user satisfaction [26][33][60][71]. Usability engineering can give insights into user motivations, characteristics and work environments to further HCI research.

Broadly speaking, there is a mutually reinforcing relationship between tasks people need to perform and the artifacts that support them [14]. Peoples’ needs and activities drive the development of systems to support them which in turn lead to the discovery of new needs and possible activities. This task-artifact cycle drives the development of computer systems and technology in general. Usability engineering works within this framework and generally follows an iterative process that includes identifying high level design goals, developing an understanding of the users, designing the interface, and evaluating the usability of that interface. There are a number of different usability engineering methodologies and techniques that cover different aspects of the usability engineering lifecycle including requirements analysis, prototyping and empirical evaluation. For example, Constantine and Lockwood’s usage-centered design is a general approach to design that focuses on how users are trying to accomplish their tasks, rather than just understanding users themselves [24]. Cooper advocates the use of personas—a fictitious exemplar user developed from actual user data that describes a user’s background, needs, goals and tasks [26]. Personas can help guide and focus development—especially if designers do not have access to actual end users.

My work, though based mainly on scenario-based design principles, draws on a number of different methods and models. It is a demonstration of how usability methods and agile methods can
be used in concert to develop usable systems efficiently and effectively and could potentially be used with other usability practices such as those highlighted above.

### 2.1.1 High-level design goals

An important early part of usability engineering is setting appropriate high level design goals to guide subsequent development. These goals are important in that they can help designers prioritize design options and make informed decisions [60]. When combined with well-defined usability metrics that are tied to those goals and measured at regular intervals, developers can get a sense of how closely their design matches up with those high level design goals.

Several researches noted that most new systems do not draw on previous research and development efforts [58][81]. This results in a large body of unrelated HCI development and research efforts with no accumulation of proven knowledge or ability to judge whether one design is actually better than another. As a result, the field of HCI lacks well defined, accepted research foci or set of common problems that other areas such as information retrieval use to advance the state of the art. This also makes it difficult for practitioners to leverage design knowledge from past efforts in developing systems or to quantitatively determine whether their system is usable or not.

Newman introduced the concept of critical parameters to HCI as a way to address this problem. Critical parameters are figures of merit that transcend specific applications and focus on the broader purpose of a particular class of technologies [59]. They can function as benchmarks that provide a direct and manageable measure of how well a design meets its purpose. They apply not to a specific system, but to a particular design problem and allow for the comparison of different designs that attempt to address the same problem. In addition, they are assessed with empirically gathered data and so are easier to monitor and affect throughout the design process. For example, one critical parameter used for vehicle design could be the mileage of the car. Several different car designs could be developed by different teams but the resulting vehicles could both be rated against that same target mileage.

Within the field of HCI, McCrickard has developed critical parameters for notification systems—systems used in dual task situations to supply users with information while minimally interrupting them from some primary task [46][47]. Instant messengers and e-mail alerts are common examples of such systems. The domain can be organized by three critical parameters that define innate aspects of notification systems. Each critical parameter characterizes the prominence of a psychological effect caused by the design. **Interruption** (I) is the reallocation of attention from the primary task to the notification in the secondary task. A **reaction** (R) is a response to the stimuli to determine whether the notification should be further pursued. Finally, **comprehension** (C) describes
the process of understanding the notification and storing the information in long-term memory. Interface developers can define design goals in terms of these critical parameters on a scale of 0 to 1 and use them as usability and performance metrics for comparison during evaluations. For example, an alarm might have IRC ratings of I:1 R:1 C:0. This indicates that the alarm should be highly interruptive so it is more likely to attract the users’ attention, support high reaction so users can respond to the notification quickly and support low comprehension since alarms do not require users to understand and remember details about alarm events.

Much of my work in usability engineering leverages the concept of critical parameters—specifically McCrickard’s use of IRC critical parameters for notification system design.

2.1.2 Scenario-based design

One established usability engineering approach is scenario-based design, a design representation-based process that uses scenarios—narratives describing users engaging in some task, 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 [14][16][71]. Claims provide compact, designer-digestible packets of knowledge ideal for use in time-critical design activities [77]. Figure 1 shows an example scenario and claim for the notification collage, a virtual notice board to allow users to maintain awareness of people they work with [31].

Scenario-based design specifies four design phases: requirements analysis, activity design, information design, and interaction design. Requirements analysis is where designers first collect information on current practices through interviews, ethnographic studies and other data gathering techniques. This information is used to construct a root concept document, which describes the overall vision for the system and stakeholder descriptions. The designers then craft problem scenarios and claims to describe how tasks are currently done and what key problems and issues exist. In activity design, designers develop scenarios and claims to describe activities and tasks the new system will support based on the previously developed problem scenarios and claims. In information and interaction design, designers determine how the activities will be supported through the information the interface provides and the interactions it supports. These phases, though defined serially, often intermingle in practice as the nature of the task-artifact cycle means that requirements and user needs may change as the system is iteratively developed. For example, information and interaction design may occur in a single iteration, followed by a usability evaluation that uncovers a new requirement and prompts the designers to reconsider the overall activity design. Specific evaluation, prototyping and requirements analysis methods are not defined and are chosen based on the nature of the system being developed and other situational factors.
Pascal, a graduate student, is working on a paper related to his research. While working on the paper, he also wishes to be informed of information that is being shared within his lab. By using the Notification Collage (NC), which runs on his second monitor, Pascal can now glance at the NC every once in a while in order to see the posted items. When looking at the NC, he visually scans the randomly placed items that are on top. As he looks at the various types of information posted, he gains an understanding of the information contained in the items that are completely visible, but does not know if the information is recent. Knowing that he must find out at a later time when the information items were posted, he returns to his research paper.

Information artifacts haphazardly posted in an unorganized fashion onto a public display for relevant information delivery, similar to postings on a bulletin board.

+ allows users to gain an understanding of an item's age/applicability with respect to the number of items that may be covering it
+ the lack of information categorization accommodates a wide range of different types of information to be conveyed through the display
- BUT overlapping items due to the lack of organization can hinder efforts to read/see a particular information item
- BUT the actual age of an item is not apparent

Figure 1. Scenario and associated claim for the notification collage. The claim highlights the tradeoffs of the bulletin board metaphor used in the system.

Carroll asserts that theory-grounded HCI research can drive innovation by expressing, testing, and reusing “falsifiable” hypotheses (or claims) about the psychological effects an artifact has on a user. Scenario-based design (SBD) is an approach to interface development, providing an inquiry method to help designers reason about elements of a usage situation and receive participatory feedback from stakeholders [71]. Using Carroll’s approach, HCI professionals and software developers conduct an explicit claims analysis in formative and summative usability engineering efforts, continuously striving to balance and validate tradeoffs of use. A claims analysis record for a single system, and the accumulation of records from multiple systems, holds valuable design knowledge that, as Carroll has argued, should facilitate component reuse, motivate further empirical research, and inspire high-impact innovative development. Furthermore, combined with the use of critical parameters, the SBD approach can help practitioners by driving the development of demonstrably more useable systems.

My work draws primarily from scenario-based design as an exemplar usability engineering process that embodies many of the key values and practices common among usability engineering processes. SBD is also used as it was developed to bridge the apparent gap between academic research in HCI and usability engineering in practice.
2.2 Emergence of agile methods

In software engineering, prescriptive process models are used to control the unpredictability and fluidity that can surround software development. Traditional software methodologies 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 [7] 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 [69]. Many of these 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 [4][29][32][69]. 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. Agile methodologies have emerged in the last decade as a way to address constantly changing requirements and other key problems including the increasing cost and complexity of software development, communication breakdowns among stakeholders, and missed schedules and budget overruns. Agile methodologies purport to address these software development problems by focusing heavily on quick delivery of working software, incremental releases, team communication, collaboration and the ability to respond to change.

2.2.1 The Agile Manifesto

One unique feature of the area of agile design is that agile practitioners are guided by the same set of high level values codified in The Agile Manifesto [6]. This set of values was developed by a group of leading agile practitioners known as The Agile Alliance to develop common ground among the different emerging agile methodologies and to make concrete the key principles embodied in all agile methods. The manifesto states that agile practitioners 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

10
Part of the reason for developing the manifesto was to make clear what the priorities of agile practitioners were and to partly dispel the notion that agile methods were undisciplined and unstructured. The manifesto states that agile practitioners find value in the items on the right, but they value the items on the left more.

This set of values was further expanded into a general set of principles to further detail the core foundations of agile methodologies [6]:

- **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.**
- **Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.**
- **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.**
- **Business people and developers must work together daily throughout the project.**
- **Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.**
- **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.**
- **Working software is the primary measure of progress.**
- **Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.**
- **Continuous attention to technical excellence and good design enhances agility.**
- **Simplicity—the art of maximizing the amount of work not done—is essential.**
- **The best architectures, requirements, and designs emerge from self-organizing teams.**
- **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.**

Generally, all agile methodologies adhere to these principles and they all relate back to the four key values stated in the manifesto. The principles themselves do not provide specific process guidance. For this, one needs to look at the specific agile methodologies that have been developed.

**2.2.2 Agile landscape**

A number of different agile methodologies have been developed around these principles that focus on different issues including programming practices, project management, business factors and
agile adoption [5][73][68][75]. This section briefly describes several agile methods that focus on different aspects of the software development process.

Adaptive software development (ASD) is based on complex adaptive systems theory and views software project teams as a complex adaptive system—namely it is composed of multiple interconnected agents and can change and learn based on previous experience. The agents consist of developers, clients and other project stakeholders and are operating within the confines of an organization, and development process to develop a product [44]. The ASD development process is centered on a collaborative learning cycle that focuses on continuous progress and a focus on delivering the working product through an iterative development cycle.

Lean software development (LD) is a set of principles and tools that software organizations can use to make software development projects more efficient and is derived from the area of manufacturing [68]. LD is different from other agile methods in that it has more of a focus on the business side of software development. It includes tools such as identifying and eliminating waste practices that do not directly contribute to the final end product and directly mapping all the steps in a development process to the value it contributes to the end product.

Scrum is an agile method that focuses more on the project management side of agile development processes rather than the specifics of software development [73]. As such, it can be used in conjunction with more development-oriented processes such as extreme programming [36]. Each team is managed by a Scrum Master, whose role is to ensure that Scrum practices are followed, act as the liaison between management the Scrum team and the customer, and to remove impediments to progress. Like other agile methods, software is developed in iterations known as ‘sprints’, which are typically several weeks long. Each sprint will involve the development of the highest priority features in the ‘product backlog’, an evolving list of business and technical functionality the system needs to support. Daily meetings, known as ‘scrums’, are brief meetings where the team can communicate and exchange information and identify and issues that arise.

### 2.2.3 Extreme programming

One of the most widely adopted agile processes is extreme programming (XP), developed by Kent Beck with the help of Ward Cunningham and Ron Jeffries [4][5]. It is derived from a number of existing software development practices such as cyclical development cycles and feature-driven development and is focused on delivering value to the customer through a flexible development process that reduces the cost of change curve [7][34]. Extreme programming largely 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. It is built around a set of
5 key values: effective *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 [5]. One can see that these values can be directly tied to the values highlighted in the Agile Manifesto. Extreme programming reduces the cost of change curve through a tightly coupled combination of planning, design, coding and testing practices. Traditionally, this process relies on the entire team—including the customer—to be at the same location, so they are able to work together most effectively.

**Planning.** Like other agile methods, XP follows an incremental development process that is divided into releases which are in turn divided into iterations [44]. This allows developers to provide demonstrable value to customers in a short amount of time and allows continuous feedback to be obtained. In release planning, project members identify system requirements in the form of stories—which describe specific features that need to be developed. These stories are prioritized by customers. The team then determines which set of stories will be developed in the next release based on customer needs and business value. In each iteration, which typically last several weeks, developers break the highest priority stories into development tasks which are estimated and assigned to the developers. At the end of each iteration, it is expected that a working piece of software is completed and tested. It is possible for stories to change or for new stories to be identified throughout the process as the software is incrementally developed and reviewed.

**Design.** At project inception, project stakeholders identify high level project goals and identify a metaphor—an overall concept of the system that is being built. This will help the team to make consistent design decisions and aide in collaborations with one another. In addition, one key practice of XP is to use the simplest design possible that fulfills a requirement. This is so development work is not wasted on functionality that ends up not being needed in the future. Developers are encouraged to refactor their code regularly so that the resulting designs are as simple and easy to understand as possible.

**Coding.** One well-known XP practice is pair programming. In pair programming, two developers work together at a single workstation whenever they are coding. One person is doing the actual coding while the other is in charge of looking at the broader design and coding issues. For example, this person might determine that the code needs to be refactored or may identify a better algorithm for solving a problem. In XP, code is ‘collectively’ owned by the team as a whole. If one programmer sees a change that needs to be made, then he has a responsibility to make it himself. This requires all programmers to be familiar with the code base as a whole.
Testing. A heavy focus on testing makes the development and design practices such as refactoring and collective code ownership possible. XP follows a test-driven development process where programmers first write the test cases for a story, before writing the code itself. These automated test cases are combined into a common test suite that includes all the tests written by all the programmers. The programmers can then integrate any changes they made into the project code base by running it through the entire test suite. This way, there is always a daily build which includes all the latest changes made to the code and that functions without identified errors.

My work leverages extreme programming because it is one of the most widely adopted agile development methodologies. In addition, a number of the practices and artifacts in XP are similar to those in scenario-based design.

2.3 Enabling agile usability

One shortcoming of agile development methodologies, and of many software development processes in general, is their marginalization of usability issues [7][65]. This is especially true of agile methodologies such as XP as they were originally developed to focus on satisfying development and business needs rather than on end user needs. Agile methods and usability practices have much in common. They both follow cyclical development cycles, are human-centered and both emphasize team coordination and communication. However, differences in the foci and goals of the two areas may cause conflicts that can hinder the development process [57]. Agile practitioners have acknowledged the need to develop systems that meet usability requirements in addition to meeting functional and market requirements and have begun to explore ways of incorporating usability into agile methods [7][20][24][28][52][65][68][74].

A combined approach is difficult because agile methods, which are incremental and iterative in nature, do not support any kind of comprehensive overview of the entire interface architecture which is an important part of making consistent and usable interfaces. Constantine advocates a combined usage-centered approach that begins with interface design, continues with existing agile software development processes and relies more on modeling than on user testing [24]. Cooper argues for a similar 'usability-first' approach where usability professionals first interact with customers to develop the interaction design before working with developers to implement it [57]. One potential problem with this approach is that the interface usability design process becomes a bottleneck in the overall development process. In addition, the reduced emphasis on end-user testing may provide insufficient feedback to drive subsequent development efforts.

One other problem with a combined approach is the communication problem between usability engineers and software engineers. Differing goals, mindsets and backgrounds can lead to
conflict about how best to manage a project. Patton has advocated an approach that works around this problem by having existing agile teams learn about and integrate usability practices into their day to day tasks [65]. One potential problem with this approach is that usability engineering is a separate discipline that is not a simple endeavor for systems with a large UI component. In addition, the additional burden of the usability tasks may be taxing on developers and slow down project velocity.

Ambler advocates a more integrative agile approach to usability engineering where just enough requirements analysis and modeling is done up-front to begin development. Subsequent user interface modeling, design and evaluation occur continuously in parallel with development [1][2][7][20][50]. Similarly, Lynn Miller et al. at Alias, suggest a methodology where software development and usability engineering proceed in parallel tracks [52]. The usability team designs the interface for the next iteration which the developers will then work to implement. Usability specialists also validate the usability of what was developed in the previous iteration. These approaches generally acknowledge software engineering and usability engineering and distinct and separate areas of expertise. In this case, communication and careful coordination are vital as agile developers and usability specialists can have differing motivations, thought processes and goals. One problem with this approach is that there is no well defined process for maintaining a consistent and coherent interaction architecture since there is minimal up-front design [57].

2.4 Chapter summary

This chapter provided an overview of foundational work in the fields of usability and agile methods. Most usability engineering methodologies are built on the same principles of understanding users, their needs and the design context and following an iterative design-evaluate development cycle. Similarly, agile methods generally follow the values and principles laid out in the Agile Manifesto. This dissertation work is focused on combining agile software development and usability engineering and is based on two specific methodologies, scenario-based design and extreme programming. My approach will be to have development and usability teams working in parallel, but to also enable the development of a consistent high level interaction architecture design through a shared design representation that is developed and maintained incrementally. It also differs from other agile usability techniques in that it brings active research in HCI and design methods in areas such as ubiquitous computing, computer-supported cooperative work, and user evaluations and modeling into practice. This will address the common criticism that HCI and usability research does not apply to practical situations—resulting in an apparent ‘theory-practice gap’—by relying on shared design targets, knowledge capture, and practical development concerns that cut across design projects.
The next chapters will detail my work in scenario-based design, the central design record, and how it is combined with extreme programming to form the XSBD process.
3 Background work

This chapter summarizes my initial work in scenario-based development, design representations and knowledge reuse [39][40][49][56]. It describes my work on the LINK-UP system, an online scenario-based toolset that supports SBD and knowledge reuse. This work led to my initial work on design representations—based on Norman’s work on the system image [63] and Carroll and Rosson’s work in scenario-based usability engineering [71]. This work and associated research studies, involving students working on semester-long development projects, provided the foundation from which my current research on integrating usability and software development methodologies is based.

3.1 Leveraging a design representation in LINK-UP

The goal of this initial work was to design and develop a tool to enable the learning of usability engineering and HCI concepts while contributing to a growing and evolving development environment called LINK-UP. It focused on teaching students and novices about analytic evaluations, a popular usability evaluation method that can be conducted rapidly and inexpensively. These evaluations were centered on the use of claims as a point of discussion of design features among evaluators and designers [40]. This work further discusses how this system supports design knowledge reuse through claims. This section is adapted from my E-Learn ’04 conference paper: From Chaos to Cooperation: Teaching Analytic Evaluation with LINK-UP [40].

3.1.1 Introduction

The field of human-computer interaction (HCI) is concerned with the development of interfaces that allow people to effectively use computers to meet their goals. This multidisciplinary field of study is difficult to teach in higher education settings and may appear to students as a seemingly random collection of topics, without the structure important to facilitate learning. Our efforts seek to design and develop a tool that will enable students to learn basic concepts of HCI while contributing to a growing and evolving effort. Key in our system are analytic evaluation methods for software interfaces, which enable students or design professionals to cost-effectively conduct a usability test in minimal time with few participants [61]. Like students of other design disciplines, interface designers encounter valuable learning opportunities when they have access to many examples of existing design products and multidisciplinary perspectives. Therefore, information from the evaluation is stored for reuse in other phases of development, and by future system developers.
With this high-level vision, our efforts target development and initial testing of such an analytic evaluation tool within an emerging web-based, integrated design environment—LINK-UP.

3.1.2 Background

Notification systems design with LINK-UP

To develop and validate our prototype design environment and analytic testing tool, we have constrained focus to a specific challenge within the field of HCI—developing notification systems. As a specific type of computer interface, notification systems typically are used for very brief information interactions, unlike most interfaces that are used for extended periods. Users value notification systems for reacting to and comprehending time-sensitive information, without introducing unwanted interruption to ongoing tasks [46][47]. Examples include news tickers, alerts or pop-up windows, automobile displays, system or network monitors, and context-sensitive help agents. We have found that notification system design problems can make very rich assignments for HCI students—requiring thoughtful consideration of multimodal design tradeoffs and information visualization techniques to effectively present interface state transitions to users [22].

LINK-UP, or (Leveraging Integrated Notification Knowledge with Usability Parameters), is an integrated design environment for notification systems that allows designers to proceed through development activities while accessing and creating reusable design knowledge with claims as the basic reusable knowledge unit—an idea first proposed by Carroll in his Task-Artifact Theory [18]. Complementary to a scenario-based design approach, the LINK-UP system provides interactive modules and tools for activities that include requirements analysis (brainstorming) with task-models, participatory negotiation with end-user stakeholders and analytic and empirical evaluation [21]. Through the progressive use of LINK-UP, developers create and reflect upon interface artifacts defined at ever-increasing fidelity. LINK-UP helps developers focus HCI concerns on the critical parameters of notification system design—controlling appropriate levels of user interruption, reaction, and comprehension, or IRC parameters [47]—with its integrated access to detailed notification design tradeoffs (or claims) established through hundreds of previous design efforts, empirical testing results, and concise summaries of psychology and human factors literature. As developers use LINK-UP, these basic research results are applied and put to the test with new design efforts, continuously advancing the state-of-the-art in notification research. To widen the scope of reuse, the knowledge contained within claims and their associated artifacts must be classified and generalized; fortunately, a schema and method for classifying claims is introduced by Sutcliffe and Carroll [77]. LINK-UP is intended to provide a framework for design knowledge reuse, coupling a claims library with a suite of design-support tools [21].
Encouraged by initial acceptance of our broad approach to computer-aided design of software interfaces, recent efforts focus on development and validation of specific modules within LINK-UP. Arguably the most critical module within LINK-UP, the work reported here relates to the Analytic Module.

**Analytic Evaluation in LINK-UP**

Generally, an analytic evaluation is a process in which evaluators (often experts) identify usability problems in a partially implemented interface by ‘simulating’ the way they think users will perform certain tasks. One of the fundamental ideas in HCI is that usability evaluations should start early in the design process, optimally in the stages of early prototyping. When critical design flaws are detected earlier, the chance increases that they can and will be corrected without requiring costly evaluation with actual users (i.e. empirical evaluation) or major code changes. The analytic evaluation module will ultimately support several different types of evaluations, such as the use of heuristics [61].

However, in this prototype, we employ a critical parameter comparison technique that contrasts the design model (expressing the designer’s interpretation of user requirements) with the user’s model (characterizing the actual user’s experience)—concepts originally introduced by Norman [63]. The Analytic Module within LINK-UP provides a tool that facilitates execution of an analytic evaluation method, records evaluation results, and provides an estimate of what the user’s model IRC parameters might be if the system were fully developed and tested with users (the analytic model). That is, to support the goal of early evaluation and introduce students to this essential practice, designers are able to get a preliminary sense of the levels of interruption, reaction, and comprehension that their system design will cause within users. To obtain the analytic model, designers prepare an overview of the key interface interactions (or a system image, which may include a system prototype description, interaction scenario, and prototype artifacts). Expert evaluators use this overview to walkthrough and analyze the interface with a series of questions established as an IRC analytic calculation method [23]. During the evaluation, experts also consider the designer’s claim statements about psychological effects that the interface will cause in users. Claims can relate to problem spaces (e.g. describing situational characteristics and requirements) and be characterized by a design model IRC, or claims can express proposed design solutions. The module assists designers with matching problem claims with design claims—a process that will assist designers later in pinpointing the underlying causes, if the design model and analytic model IRC values do not match after the expert evaluation.
3.1.3 Analytic module development

We began brainstorming about the design of the analytic module by considering all of the broad activities that users would need to accomplish as they proceeded through an analytic evaluation. Two important realizations emerged: first, the system would be used by two distinct types of users; second, the module would need to support several activities beyond just the evaluation execution. Each of these ideas is elaborated on below. Most of our early design work was conducted with rapid prototype development software that allowed quick storyboard sketching and linking of steps within the module (Figure 2). In the early process, we were unconcerned with the specifics of information design—we focused solely on ensuring that all essential processes and steps were included. We received frequent feedback on several iterations of our rapid prototypes from members of our larger research group. As our ideas solidified, we emerged with a coherent task analysis and screen-flow for the analytic module.

There are two types of users that will use the Analytic Module, designers and evaluators. Designers are the individuals who will develop the system to be evaluated using scenario-based design and LINK-UP’s claims library. Evaluators are the individuals who will be assessing the system’s usability. Ideally, they would be experts in HCI and be familiar with scenario-based design claims analysis [71]. According to our needs analysis, we determined that the analytic module should be divided into three sub-modules: Manage System Image, Evaluate System Image and Review Evaluation Results. The three sub-modules correspond to the basic sequence of tasks (Figure 3). The designer’s and evaluator’s tasks will be described respectively below.
Figure 3. Basic sequence of tasks of the analytic module of LINK-UP.

First in the task sequence, supported by the Manage System Image module, designers will either add a new system image into the module or edit an existing system image. To create a new system image, the designer will need to select an ongoing design project. Each design project will be associated with a list of problem claims developed during the requirements gathering and participatory negotiation phases of the LINK-UP system. The designer then links necessary design claims to each problem claim. Each design claim should solve part of or the whole problem addressed in the problem claims. Design claims can either be newly developed by the designer or be reused, drawn from the claims library. Artifact representations are also uploaded to describe the system image. An artifact representation shows some aspect of the system that is being designed. They can be pictures, screenshots or movies of the system interface. Before this process, the designer would have already established a design model IRC, which represents the intended IRC value for the system being designed. Once this process is complete, the system image will be ready to be evaluated. Designers can also edit existing system images after creating them in response to evolving requirements or to account for system redesign decisions.

Second, through the Evaluate System Image module, HCI experts can analytically evaluate system images. The analytic module will step the evaluator through the evaluation method and record the results of the assessment. During the process, the evaluator would first become familiar with a summary of the system image to develop an overall understanding of the notification system design. This is accomplished by browsing through the system description, the interaction scenarios, the design model IRC values, and the prototype artifacts. Then the evaluator steps through each problem-design claim pair and rates how well each design claim addresses the associated problem claim on a four-point scale (major flaw, flaw, minor flaw, or no problem). This rating is based on how well the upsides (positive psychological effects) expressed by the design claim either:

- Support the upsides of the linked problem claim
- Mitigate the downsides of the linked problem claim or other design claims

Evaluators provide comments for each claim pair to explain their ratings. At the end of the evaluation, the evaluator runs the IRC equation algorithm to get the estimated IRC value for the whole system [23].
As the final step in the task sequence, in the Review Evaluation Results module, designers will be able to review the results of evaluations that experts have run on their system images. At the start of this interaction session, the designer will see overall evaluation comments and will be able to compare the design model IRC and estimated IRC for their prototype. This gives the designer a sense of how well the system might fulfill its most critical user goals, if fully developed. They can then view the ratings and comments for each claim pair to elaborate on specific, causal design features. The designer can use the ratings and comments to improve their designs. They can also revise the problem-design claim linkages, or even modify their original design claims. Since the claims and their evaluation results are archived in the claims library and accessible beyond the single project, future developers will be able to benefit from this stored knowledge.

3.1.4 Pilot testing evaluation methods

We conducted a three stage formative empirical usability evaluation of the analytic module—a process in which potential users actually used the prototype to accomplish real tasks. This evaluation was conducted after the development of the module prototype to get feedback from actual designers and study how it is used in a realistic situation [71]. The evaluation had two goals, to establish whether:

- Designers can use the module to help identify and resolve design problems
- Through use of the module, could designers learn more about claims-based usability engineering

The first goal arises because an overall objective of the LINK-UP system is to support the usability engineering process for notification systems and enhance HCI education activities [21]. The second goal arises because many initial users, especially those in an academic setting, are unlikely to be experienced with claims and scenario based design. The analytic module addresses the user’s tendency to learn about a new tool through active and informal, exploratory use [19].

In our pilot testing, we were assisted by seven participants who were all undergraduate students in a semester-long undergraduate research seminar, focusing on design and evaluation of notification systems. The students had spent recent weeks individually designing a notification display that helps users monitor and react to constantly changing airplane ticket prices. At the start of our pilot testing, each student had a semi-functional prototype notification system to support this typical notification task. The evaluation was run over a period of three weeks, with one hour-long session held each week. To prepare for the evaluation, participants added to the claims library their problem and design claims. In Session 1, the participants created the system image for their prototypes using the Analytic Module. The participants had the option of linking their own (new)
design claims or reusing existing design claims to their problem claims. After the system images were created, each participant completed a questionnaire containing multiple-choice statements (rated on an eleven-point Likert scale) and general questions that required written answers. This questionnaire assessed the designers’ opinions about how well the system image represented the prototype design. In Session 2, each participant evaluated the system image of a prototype developed by another participant in the last session. This was also followed by a questionnaire that gathered feedback on how effectively the participants were able to evaluate the prototype using the module. In Session 3, the participants reviewed the evaluation results for their system images and completed a form to express comments and reactions to all information resulting from the evaluation (quality of evaluator feedback and utility of the general approach). This session was followed by a questionnaire that focused on whether the evaluation gave them useful design feedback and whether evaluators identified any problems with the claims. A detailed overview discussion of the study results is available here [40].

3.1.5 Summary of study findings

Overall, we can report that the designers and evaluators were able to understand and use the prototype of the Analytic Module as it was envisioned. The pilot testing showed that distributed project work is feasible with the support of at least this module of LINK-UP. We were also pleased to observe a few cases of design knowledge reuse, and we are confident that more cases will be apparent as the tools are more fully integrated in design practice and as the claims library content develops further. To summarize the major findings:

- A prototype representation (system image), expressed in terms of problem and design claims provides enough information for knowledgeable HCI persons to quickly evaluate prototype designs.
- The Analytic Module can help designers identify and resolve problems with their prototype designs in terms of design claims and how they address both the upsides and downsides of problem claims.
- The Analytic Module can help students learn about scenario-based design with claims analysis by giving targeted feedback through evaluations that can uncover misconceptions about claims and relationships between problem and design claims.

The results of our online evaluation tool demonstrates its potential for teaching HCI design concepts such as claims analysis and knowledge reuse to students in a distributed and collaborative environment. However, based on participant feedback, the Analytic Module should separate the process of rating the quality of claims from the process of rating the quality of the design they
represent. Furthermore, supporting multiple evaluations for a single system image would strengthen the believability of the ratings.

### 3.2 Bringing design representations to the forefront

Our continuing work in developing LINK-UP, an integrated design and reuse environment, addresses the need for proven, dependable engineering processes for interface development [39]. It suggests that a better understanding of the system image is key to the successful evaluation of design prototypes, and an aide in applying knowledge from the repository. This section describes our work to enhance LINK-UP by developing and augmenting the system image to make it the central communication point between different stages of design and between different stakeholders. We report on a study of the modified task flow that demonstrated the value of the system image within a broader design context. Overall, our findings indicate that the effective creation and use of knowledge repositories by novice HCI designers hinges on successful application of existing HCI design concepts within a practical integrated design environment. This section is adapted from my ACMSE ’05 conference paper: Image is Everything: Advancing HCI Knowledge and Interface Design Using the System Image [39].

#### 3.2.1 Introduction

This work is a continuation of our effort to develop a structured, methodical way to both apply and further develop HCI knowledge through LINK-UP, an integrated design environment that supports a scenario-based usability engineering process centered on the use and development of a knowledge repository. The overall goal of LINK-UP is to validate the use of a reusable knowledge repository as a basis for developing a more principled approach to design that can build on past efforts. Key in creating this is adequately capturing the vision of the designer and relating it to the experience of the user to enable reflection and iteration on the design. This will in turn support the use and creation of reusable knowledge.

In our preliminary work, we integrated Norman’s ideas into the analytic module to allow HCI experts to evaluate nonfunctional prototypes using a digital representation of a system image [63]. An initial study confirmed that the system image supported effective analytic evaluation of initial design prototypes, but also suggested that it was instrumental in the application of information from the knowledge repository. We worked to redesign the initial architecture to further develop our own interpretation of Norman’s system image, as well as to apply other concepts derived from HCI so that LINK-UP better supports interface design and the generation of reusable design knowledge. We conducted a study of the new task flow with a group of novice HCI designers that demonstrated the
value of the system image as a central part of the LINK-UP system and its effectiveness at supporting interactive system design. However, participants encountered problems during the design process caused by differing understandings of key HCI concepts that were integrated in LINK-UP.

![Figure 4. Norman’s conceptualization of the system image. The designer captures system intentions in a prototype, along with supporting documentation, enabling the study of user interaction with the system.](image)

### 3.2.2 Background

Norman established the idea that governing the usage experience is the consistency of two conceptual models—the *design model* held by the designer and the *user’s model* based on the user’s understanding of the system [63]. Each model can be analyzed as stages of action, which describe the cyclical evaluation and execution of tasks across the *Gulf of Execution* and the *Gulf of Evaluation*. To facilitate a user’s evaluation and execution of tasks, designers must develop conceptual models as they would develop the scaffolding of a bridge. Several factors contribute to each of these conceptual models. The design model should be inspired by a requirements analysis, including consideration of a user’s background, situational context, and task-oriented goals. This model expresses the designer’s understanding of user needs and is a representation of the intended functionality for the system. The user’s model is formed by the user’s understanding of the *system image*, the physical system and its documentation (Figure 4).

The key idea we continue with is that Norman’s view of the role of an interface designer is to develop the system image so that the user’s model and design model are compatible. Carroll’s work focuses on a scientific approach to research and interface development that drives interface innovation and HCI research through an approach like his scenario-based design process. Through the development and sharing of narrative descriptions of users solving problems with designed systems, designers are able to create the scaffolding across Norman’s Gulfs—and develop systems with design-
user’s model compatibility. The convergence of Norman’s ideas with Carroll’s, provides the theoretical basis for our project: The iterative process of gauging critical parameters, embodied in design artifacts and expressed with claims, should guide an evaluation of the system image and provide indices for reusable design knowledge.

### 3.2.3 Enhancing LINK-UP

In the previous study, the participants failed to effectively leverage critical parameters as a way to guide the analytic evaluation process and the subsequent redesign of their systems. This was because students could not see the connection between the system IRC values, which describe the IRC values for the entire system, and how they could be applied to design decisions involving individual features. We began redesigning LINK-UP to address this shortcoming and further investigate the incorporation of existing HCI concepts so that it better supports design processes and the creation of reusable design knowledge.

To this end, we incorporated Carroll’s interpretation of the stages of action [71] within the LINK-UP development process. As a result, designers would express requirements for each stage in terms of the IRC critical parameters. This was done to provide additional guidance to designers in developing and applying claims and to better support the use of critical parameters within the design process.

Within LINK-UP, the system image consisted of the scenarios, claims and IRC values developed for the system in addition to associated screenshots and drawings depicting the system. The task-flow was modified such that the system image would act as the central component of the LINK-UP architecture through which the different modules would interact. For example, the result of an evaluation from the empirical evaluation module might suggest some required design change. The designer can then run a participatory negotiation session with end users using the system image as the communication point between them. Changes that are made to the prototype design can be reflected in the system image, which can then be used by any of the other modules in the next design iteration. LINK-UP therefore better supports the rapid, iterative development cycles common in software engineering and which are encouraged in scenario-based design [71].

### 3.2.4 User study description

We conducted a study to validate that the central role of the system image, combined with the addition of IRC values within a stage of action framework, supported the design of notification systems. We also expected to encourage the development of claims so as to contribute to a knowledge
repository. In addition, the study was intended to highlight difficulties novice or inexperienced users may have in using the LINK-UP system to design an interface.

Participants for this study were nine computer science graduate students in an upper-level graduate HCI course. Participants had little to moderate amount of HCI knowledge and had little to no experience applying that knowledge to interface development. The study was run as a series of two project-based homework assignments. For the first assignment, participants were asked to define aspects of a problem situation to motivate the design of a notification system through the development of problem scenarios, claims and high level goals in the form of critical parameters. This represented the design model for their prototype.

For the second assignment, participants developed a scenario describing the use of their prototype, and design claims that were linked to each of the problem claims they previously developed. The nonfunctional prototype, which consisted of a series of screenshots or sketches, combined with the scenarios and claims made up the system image for the design. Third, participants exchanged system images and evaluated each other’s prototypes. The written usability reviews formed the user model for each design. Last, participants were asked to compare the user and design models for their prototypes, and look for possible improvements to their design based on the results. They then completed a questionnaire that focused on how they used the system image how it helped in evaluating the system. A detailed discussion of the study results is available here [39].

3.2.5 Summary of study findings

The study demonstrated the value of making the system image a central part of the LINK-UP system and making it the communication point between the designers and evaluators. The scenarios and claims, which were sorted by stage of action, allowed evaluators to give targeted feedback to designers that helped them identify areas for improvement in their prototypes. The system image supports a similar kind of interaction in other stages of the design process such as in participatory negotiations between designers and clients. This is because it encourages discussion of the most important features of a design, using a common language that is understood between all parties. This common language, or common ground, is critical for efficient communication between parties [53].

In addition, the majority of the participants agreed that the system image would be useful later on in the design process even if a functional prototype were available, because the system image would always present only the most important aspects of the functional prototype in the form of claims and critical parameters. In addition, the system image supports efficient changes to both the design and to the requirements of the system since it ties the design and user model together within the stages
of action. This is important in practical, iterative design processes where changes may affect any aspect of a system’s design.

The key problems encountered during the evaluation process were caused by differing understandings of the critical parameters and their application to the stages of action. Scenarios and claims are largely written in the form of narrative text, which allows for explanations and elaborations. Indeed, the claim itself elaborates on design decisions presented in the scenarios and in the prototypes. On the other hand, critical parameters are quantitative in nature and their effectiveness is dependent on a universal understanding of what they mean and how they should be used. This common understanding was not established between the participants in this study. As a result, the differing interpretations rendered the IRC values within the stages of action ineffective. Instead, participants relied primarily on the prototype screenshots, scenarios, claims and on communications between the designer and evaluator.

Overall, we can report that the system image, acting as a central component of design, supported the design process presented in LINK-UP. To summarize the major findings:

- A design process centered on the system image, which represents the current design prototype in terms of scenarios, claims and critical parameters, allows designers to iteratively improve designs by having evaluators give targeted feedback.
- The system image supports communication between different stakeholders (designers and evaluators) by providing common ground based on the critical aspects of design.
- The ability of novice HCI designers to use critical parameters and stages of action within a design and evaluation environment depends on a process or tool that supports the development of a common understanding of those topics within a realistic situation of use.

### 3.3 Other preliminary work

Other preliminary work has focused primarily on the development of LINK-UP and how it supports reuse. “Use and Reuse in Information and Interaction Design” describes the use of claims as a knowledge capture mechanism [49]. It focuses on how claims are created, how they interrelate with each other and how they can be reused across different design projects. This work provides justification for the use of claims within LINK-UP. “From Personas to Design: Creating a Collaborative Multi-disciplinary Design environment”, describes how personas were used in developing the next iteration of LINK-UP [56]. Personas are descriptive models of target user groups
derived from collected data from interviews and observations. We developed four personas to target the next version of LINK-UP to: the notification system designer, the HCI researcher, the psychologist, and the software project manager. This reflects our growing need and interest in developing a way to address the needs of both researchers and practitioners.

### 3.4 Chapter summary

This chapter summarized my preliminary work in scenario-based design, design representations and knowledge reuse that was completed through my work in LINK-UP. The primary purpose of developing LINK-UP was to develop a framework to support design knowledge capture and reuse by incorporating it within the development process itself. My own work focused more on aspects of the development process itself, particularly analytic usability evaluations and the use of the *system image*, a design representation based on work from Don Norman, John M. Carroll and William Newman [14][59][63].

This work highlighted how a design process centered on the system image allows designers to iteratively improve designs through targeted feedback based on specific claims. It also demonstrated how the use of a common design representation such as the system image can help designers and evaluators by providing common ground through which they can communicate more effectively. These results, along with subsequent requirements gathering efforts [56] were used to drive the next version of LINK-UP.

Overall, this work served as a foundation for my subsequent work in combining usability engineering and agile software development processes using the central design record, a design representation that evolved directly from the system image and that was incorporated into the next version of LINK-UP. This subsequent work, described in chapters 4 and 5, is a departure from the initial focus on reuse of design knowledge—but is a direct outgrowth of and remains complementary to that work.
4 Completed work: agile usability with the CDR

This chapter summarizes how the shared design representation known as the central design record was developed and integrated into the XSBD agile usability process [38][41][42]. The CDR was initially added to the LINK-UP system as the main design representation of the scenario-based development process. It describes how the CDR was used by student developers in designing systems and the problems and issues that became apparent. It then describes how the CDR was modified to act as a central design representation that would be shared by different stakeholders in an agile software development project—with a focus on usability engineers and software developers.

4.1 Defining the central design record in LINK-UP

This work is the culmination of my early efforts at developing a usability engineering process and tool that is based on the scenario-based development process [42]. It develops and defines the main design representation used in the approach—the central design record (CDR). The CDR is a design representation meant to prevent breakdowns occurring between design and evaluation phases—both within the development team and during design knowledge reuse processes. The CDR is intended to make designs coherent and understandable, thus supporting a principled, guided development process critical for student developers. This work presents the first extensive usage evaluation of LINK-UP and the CDR. This section is adapted from my IJITSE journal paper: Understanding Usability: Investigating an Integrated Design Environment and Management System [42].

4.1.1 Introduction

While previous papers have reported on the design of LINK-UP [21][66], this paper introduces the central design record (CDR), a design representation that makes explicit where and how handoffs—such as the one between evaluation results and the subsequent redesign of a system—occur in the development process and highlights design decisions that need reconsideration during the next development iteration. It is a structured organization of design knowledge (derived from the knowledge library or newly created) that describes critical features of an interface, what problems the features address, and how evaluation and collaborative design information relates to and affects those features. In this way, it makes explicit the relationship between design goals and the developed interface. For instance, the CDR allows developers to determine links between identified problems and how and an interface design addresses them, or to see relationships between evaluation data and
different aspects of the interface. Student designers will be able to see how different design techniques affect designs and why those techniques might be used.

This section summarizes results from three case studies that illustrate how novice designers used LINK-UP to engineer interfaces. Building on past experiences with knowledge repositories and principled process-oriented development, LINK-UP serves as a culmination and realization of the prior work of McCrickard and Chewar [21][23][46][47]. The results suggest that a design knowledge integrated development environment (IDE) that incorporates the processes and principles of the central design record, can help novice developers apply a design methodology to develop interfaces in a guided, methodical fashion and avoid process breakdowns. Furthermore, there is evidence that, through the use of the CDR, iterative application and verification of reusable design knowledge—important in the education of novice interface developers—is made practical.

4.1.2 Motivation and related work

Various methodologies and techniques within HCI have identified problems and issues that need to be addressed in a usability engineering process. In this section, some of the foundational ideas that motivate the development activities supported by LINK-UP and the central design record are presented.

The creation of a detailed design representation in SBD is imperative to the design process, allowing explicit analysis of the new system in its anticipated context of use. The designer’s goal is to complete this design representation with as much detail as possible. With a well-defined design representation, the reuse of design knowledge can prove to be immensely helpful as they are more likely to fit into the structure of the design representation and contribute to the overall design. Being able to effectively store and reuse design knowledge is an active area of study for design domains [30][45][76][77]. For example, the software engineering community has long advocated reuse of both code and general code architecture solutions through patterns. Within HCI, Sutcliffe and Carroll worked on a framework for documenting and organizing claims in a knowledge repository, although as of yet they have not developed the actual library or tools to support claims reuse processes [77].

Another important consideration in usability engineering is to support communication among the different groups of stakeholders that may be involved in a development process. Given the interdisciplinary nature of usability engineering and HCI, people from different backgrounds may not have an easy way to discuss and reflect on designs. Borchers advocated the use of formally defined patterns to support communication among stakeholders [10]. In a similar vein, Borchers and Erickson have both advocated the use of patterns and pattern languages as a way to support cross-disciplinary
discourse [10][27]. Sutcliffe pointed to structured claims as a way to delivering HCI research knowledge to practitioners, giving them the ability to communicate through claims [76].

4.1.3 LINK-UP

Based on these prominent ideas and implications emerging from HCI research, requirements for LINK-UP were derived. Performing a decomposition of the overall goal of providing useful support for iterative application and verification of reusable design knowledge, the following focus points with respect to the development of design knowledge IDEs are derived:

1. IDEs must support design goal formation and facilitate continuous estimation of design progress through comparison between design goals and resulting design artifacts.
2. To effectively guide specific, incremental design improvements, the system should help developers craft a design representation that is sufficiently detailed to focus evaluation activities.
3. IDEs should facilitate communication efforts among stakeholders around the design representation and its resulting development and evaluation.
4. An interface development support system must be flexible enough to support design and evaluation activities.

With the four focus points in mind, the goal was to create a system that supports the design of notification systems through the use of critical parameters and Carroll’s notion of claims from scenario-based development. LINK-UP consists of a design knowledge repository and modules in which design activities are carried out. The repository, or the claims library [66], contains claims related to the notification system domain. Designers can search for reusable claims applicable in their own designs by using various searching or browsing features [77]. This basis, a structured collection of claims, supports knowledge sharing among designer communities interested in the domain. The central design record—developed and maintained using web-based design modules—allow designers to organize and integrate knowledge from the claims library as they develop their systems [39][40]. The central design record is based on Norman’s theory of action, specifically his concept of a system image and the need to match designer and user’s understanding of a design (Figure 4). The CDR makes connections between the designer and user models explicit, highlighting areas for improvement and gaps in the design that can be addressed in subsequent iterations using the claims library as a primary source of knowledge. Figure 5 shows how the CDR and claims library are used to iteratively develop interfaces.
Figure 5. LINK-UP's iterative design process and CDR development. Designers start at the center identifying requirements and a target IRC goal. Design iterations through the CDR include design claims which are tested through evaluations leading to convergence of the design and user models.

4.1.4 The claims library

The basic structure of a claim in the library consists of a feature and a list of upside and downside tradeoffs. This structure is extended with additional information in LINK-UP. Each upside and downside is supported by rationale either summarizing results of an observational study performed by the designer who created the claim or providing references to published research supporting the particular tradeoff. An attached scenario provides context for the claim by describing a task in which the claim can come into use. As a whole, the claim is also assigned an IRC value, the specific critical parameters developed for notification systems, allowing designers to discuss how applicable the claim may be to the overall design goals of their system. Such assignments are critical to integrating the concept of critical parameters into design, providing a base upon which claims can be evaluated. The complete structure of a claim in the library allows designers to create sufficiently detailed design representations through collections of claims.

The claims library forms the core repository of design knowledge available to LINK-UP users. Design processes to guide developers and support their use and reuse of claims are encapsulated in two different modules: the requirements analysis module and the central design record module [39]. Guided by these modules, developers construct and manage the central design records for the
interfaces they develop. These consist of an organized set of scenarios describing different usage situations, claims related to specific features in the interfaces, and design goals defined in terms of IRC parameters. The CDR acts as a living representation of the design and is used to guide all stages of design including collaborating meetings, evaluations and design improvements.

4.1.5 Requirements analysis module

The requirements analysis module serves as an introductory module for designers, and is where they first determine design goals and establish the problems that must be solved. By design, the module is organized into a series of detailed, structured steps (Figure 6). This module introduces many of the important concepts and processes within LINK-UP as novice designers make a first pass at defining system requirements before beginning development work in the central design record module. Subsequent changes to requirements or design goals are done in the more open ended CDR module. Designers are guided through the process of defining a problem scenario and problem claims.

The problem scenario gives insight into the important problems, providing motivation for a new design through a portrayal of current practices. These scenarios form part of the CDR and are later linked to design scenarios, which describe how the newly developed interface is used. This relationship—similarly defined between problem and design claims, allow developers to see connections between current practice and how their design affects and improves upon it. Designers also define design goals in terms of IRC critical parameters [21].

Figure 6. Part of the requirements analysis module. The progress/navigation bar is always visible on the right side of the screen to show designers where they are in the process.
4.1.6 Central design record module

Once developers have defined the problem situation and defined design goals in the requirements analysis module, the designer moves into the central design record module. The CDR module is the primary tool through which users develop and manage their designs. Users develop their CDRs by recording design scenarios and claims about the new system, relating them to the defined problem scenarios and claims, and iterating on them as they develop their interfaces. This module is more open-ended and less structured than the requirements analysis module to give users the freedom to change their CDRs as they develop and refine their designs.

Figure 7. Part of the central design record module. Users create a set of scenarios and corresponding claims for each supported activity. Areas of the CDR module can be accessed in any order.

Designers are first expected to create activity scenarios for each main task they have identified for their notification system. An activity scenario describes the high-level purpose and actions that are to be carried out in a main task [71]. Once the scenario is written, designers begin a claims analysis process for the scenario in which they gather claims for the activity scenario and establish relationships to the problem claims identified in the requirements analysis module. Just as in the requirements analysis module, the designer can either search for a claim or create a claim. The process of gathering claims within the module supports further claims reuse for designs and encourages the creation of more claims when none are found. A similar process is again followed for information scenarios, scenarios depicting the information a user will encounter during the task, and interaction scenarios, scenarios describing the specific actions the user will take.
The ordering of steps in the CDR module is very fluid. A specific process, as opposed to the requirements analysis module, is not imposed upon the designer (Figure 7). Many of the steps in the requirements analysis module, such as scenario decomposition, are excluded as it is expected that developers have internalized and understand these aspects of the design process at this point. Designers can switch between working on various parts of their design at any point in time. One can always choose to start creating a new task or to continue developing a previous task. This increases the flexibility of the module and permits the designer to revisit certain parts for redesign as a result of evaluation results.

The various portions of the module support the creation and maintenance of the CDR—a detailed representation of a design that can be used for evaluation purposes. For example, the breakdown of tasks in terms of the stages of action gives designers a chance to evaluate when they are nearing completion of a certain task, but also gives evaluators another perspective on how a certain task is being supported within a design. A breakdown of the design in terms of claims shows the links between prototype features and their corresponding tradeoffs encapsulated by the claims. Imposing such structure on the design in the CDR module forms a gateway to facilitating communication. Stakeholders can discuss certain parts of the design with an established common ground that focuses on specific parts of the module, helping both designers and evaluators reach consensus.

4.1.7 Case study discussion

This section summarizes results from three case studies of design projects developed using LINK-UP over several development iterations are presented. Students in an undergraduate Human-Computer Interaction course at Virginia Tech used the IDE in a semester-long project to develop notification systems. Navigation-assisting notification systems with a focus on “off-the-desktop” systems were chosen as a general theme for all the projects. Each project group consisted of 4-5 students. They reported their progress in a series of project documents written at each stage of development. These reports were designed to elicit feedback related to both the system they designed and the process and tools they used to design it. Six reports were written in total by each group, corresponding to requirements analysis, an initial design, an analytic evaluation, a redesign, an empirical evaluation, and a final concluding report. Both the strengths of this approach and areas for improvement are discussed, summarized in Table 1, with respect to the four focus points. Detailed results from the case study are available here [42].
Table 1. Summary of strengths and weaknesses of LINK-UP derived from three development case studies.

<table>
<thead>
<tr>
<th>Key focus points</th>
<th>Case Observations by Design Phase</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE supports goal formation and facilitates design progress estimation</td>
<td>Requirements &amp; Initial Design</td>
<td>Analytic evaluation</td>
<td>Redesign</td>
</tr>
<tr>
<td>(+) Generated IRC is close to expected values [case 2, case 3]</td>
<td>(+) Critical (IRC) parameters focus evaluation on non-trivial aspects of design [case 1, case 2]</td>
<td>(+) IRC parameters focus redesign efforts on critical design features [case 1, case 2]</td>
<td>(+) Evaluations focused on specific features verify hypotheses made in claims [all cases]</td>
</tr>
<tr>
<td>IDE guides design representation development to support evaluation activities</td>
<td>(+) CDR focuses on specific design features [all cases]</td>
<td>(+) Redesign efforts focused on features defined to be most important by developers [case 2, case 3]</td>
<td>(+) Claims analysis based on evaluation results guide redesign [case 1, case 3]</td>
</tr>
<tr>
<td>IDE facilitates communication among stakeholders around design concerns</td>
<td>(+) Paper prototype helps make design less abstract by connecting design rationale to interface [case 1, case 3]</td>
<td>(+) CDR provides tangible design model to support discussion of interface between developers and evaluators [all cases]</td>
<td>(+) Claims focused on analyzing design tradeoffs [case 1, case 3]</td>
</tr>
<tr>
<td>IDE is flexible enough to support iterative design evaluation activities</td>
<td>(+) Scenario breakdown helps to find claims [case 3]</td>
<td>(+) Hard to manage comprehension large # of claims and scenarios in CDR [case 1, case 2]</td>
<td>(+) Claims format aided in analyzing design tradeoffs [case 1, case 3]</td>
</tr>
<tr>
<td>(+) Requirements process constraining &amp; tedious [case 1, case 2]</td>
<td>(+) Claims search supported by critical parameters aids reuse [all cases]</td>
<td>(+) Untested or contested solutions can be deferred until empirical evaluation [case 1, case 2]</td>
<td>(+) validated claims can be updated with empirical data [all cases]</td>
</tr>
</tbody>
</table>

The critical parameters of the notification systems, embodied in the system IRC values, proved to be a valuable guide in supporting the first focus point: IDEs need to support design goal formation and comparison between design goals and the resulting design artifacts. The case studies demonstrated the importance of integrating critical parameters into LINK-UP and guide reflection and iteration on the design through the CDR. They proved to both guide design activities and estimate design progress based on evaluation results.

The case studies also suggest the second focus point—IDE aids in design representation formation to focus evaluation activities and specific, incremental design improvements—is also supported. The CDR focused evaluation and redesign activities on specific aspects of the interface and encouraged careful consideration of current practices while developing the new system. Planning empirical evaluations partly around individual claims also supported the second focus point. Failures
in the defined tests could be linked directly to design decisions—expressed in claims. Developers then knew where to focus prototype redesign efforts. The case studies demonstrate the value in supporting incremental improvements through the tight coupling of design representations with evaluation data.

LINK-UP, and its implementation of the CDR, also supports the third focus point—an IDE needs to support communication among stakeholders around the design representation’s development and evaluation. This work shows that by acting as a communication point between developers and evaluators, the CDR encourages discussion and evaluation of the interface design.

As the developers used the LINK-UP system itself, strengths and limitations of various parts of the IDE became apparent. Thus, the fourth focus point—IDE is flexible enough to support iterative design and evaluation activities—is partially supported. The relatively linear, guided process in LINK-UP, particularly in the requirements analysis module, was helpful to novice developers in identifying problem scenarios, claims and in defining design goals. However, all groups found this process to be tedious and restrictive to varying degrees. Furthermore, several groups noted the difficulty they had in managing and reviewing the large number of scenarios and claims through LINK-UP. The size of the CDR may cause other issues in the design process such as in analytic evaluations where expert evaluators may have problems reviewing and making sense of all of the CDR information.

4.1.8 Summary of study findings

Numerous usability methodologies and techniques exist to support the development of computer systems and interfaces. However, it is not always clear to novice developers how to use these different processes and techniques in a coherent design process. Through the case studies this research demonstrates that a design knowledge IDE centered on the central design record can help novice developers make connections between requirements data, design representations and evaluation data and better understand how to leverage that information to incrementally improve designs in an iterative usability engineering process. It is also shown that the use of CDR supports the application and verification of reusable design knowledge for novice developers.

Based on these results, the following guidelines are derived for design knowledge integrated development environments for use in educational settings that incorporate a module patterned on the CDR:

- Persistent design representations should support multiple or ‘current’ perspectives to direct development efforts on salient design concerns and streamline development processes.
• Design rationale should be tightly coupled to evaluation data to show novice developers where to direct redesign efforts and support validation of design knowledge for future reuse.

• Design representations need to be easily understandable, with goal states stated in unambiguous terms, perhaps through critical parameters, to support collaborations among novice developers and other project stakeholders.

• Design knowledge IDEs should support, but not require, guided processes to aid in knowledge capture, knowledge use and goal formation so that they support developers with varying levels of knowledge and expertise.

4.2 Defining the XSBD process around the CDR

The growing importance of computing systems in everyone’s daily lives has made software development an inherently multidisciplinary endeavor [62][64]. My previous work with LINK-UP and the CDR demonstrated the potential for a reuse-enabled integrated design environment in guiding novice developers through the interface design and evaluation process [39][40][42]. However, that work was not sufficient to enable usability engineers to work effectively within a multidisciplinary development team. In practice, one needs to determine how usability specialists can work within a team to develop systems in ways that can best leverage the perspectives, practices and knowledge bases of people from different areas. Software engineers, who focus more on the design and implementation of software systems, and usability engineers who focus more on the interface design for end-users, are two areas of design that have not traditionally worked well together. This is especially true of agile development methodologies, which have emerged in the last decade as a way to address problems associated with software complexity and constantly changing requirements. This work leverages the lessons learned in my previous work [39][40][42], including the use of design representations such as the CDR, and how tools such as LINK-UP should be designed, to develop practices and tools to support agile, multidisciplinary teams in developing usable systems in an efficient manner [41].

The differing goals and motivations of practitioners in software and usability engineering combined with the myriad techniques and methodologies in each leads to tensions that need to be addressed in any development process that draws on both. In this work, we probe these tensions by looking at prominent development practices in agile software development and usability engineering, namely extreme programming (XP) and scenario-based design (SBD) respectively, and explore how they can work together in developing usable software systems efficiently. Scenario-based design and extreme programming are built on similar foundations. Both support iterative development, are
human-centered and emphasize team coordination and communication. However tensions between the two approaches need to be addressed for them to work together effectively. The XSBD process supports the best practices of both processes while mitigating the tensions between them. The key features of the process are defined below. Development oriented practices such as pair programming, unit testing, code refactoring and continuous integration are unchanged. This work will focus on how to integrate SBD and XP so that usability goals can be met without violating key agile development tenets as laid out in the agile manifesto [6].

This section is adapted from my Agile ’07 conference paper: Towards Extreme(ly) Usable Software: Exploring Tensions Between Usability and Agile Software Development [41].

4.2.1 Interface architecture design

Development occurs in short iterations (2-4 weeks), beginning with an abbreviated requirements analysis process (Figure 8). In this process, the interface design—embodied in the CDR—is developed using a depth-first approach and is continuously improved and evaluated throughout the development process. This feedback loop is achieved by defining critical parameters/usability goals at the start of the process, and then using the CDR to identify critical areas of the design that need to be evaluated. Claims describe these critical areas—highlighting high-risk factors and assumptions that are tested at the end of each iteration.

Software development and usability engineering will generally occur in parallel. A successful development effort hinges on proper balance between development concerns of different stakeholder groups. The CDR acts as a shared design representation that allows software developers and usability engineers to have a shared understanding of the design that allows them to make appropriate design
tradeoffs based on constraints (e.g. usability problems, technical limitations, scheduling problems). It will be a focus point of review/planning meetings that occur at the beginning of each iteration and release cycle.

The central design record (CDR) is the main interface design representation in this process and is conceptually similar to previous efforts [39][40][42]. It consists of the set of scenarios describing different usage situations, interrelated claims describing specific features in the interfaces, and design goals. Design goals are stated in terms of critical parameters which are measures of performance used to determine how well a design serves its purpose [58]. It is used to guide all stages of interface design in the XSBD process. Task analysis and story elicitation typical occurs first and leads to scenario development but the reverse can also happen. The CDR allows developers to systematically improve the interface during the development process because it stores the rationale for the design decisions within an organized set of claims called the claims map [78][80] (Figure 8). This makes design decisions explicit and highlights important relationships between different parts of the interface.

The challenge is to develop, maintain and make use of the CDR within the tight time constraints of the XP development cycle. In the XSBD process, the CDR will be expanded as development proceeds incrementally. Developers will maintain a consistent overall view of the interaction architecture and the activities it supports by continuously reviewing and updating it throughout each iteration.

4.2.2 Collaboration through the CDR

As a record of design decisions made to the interface, the CDR also acts as a communication point between and among developers, evaluators and clients. Previous studies have shown that the CDR helps developers communicate design decisions to other stakeholders and facilitates the resolution of design issues uncovered in usability evaluations [39][40][42]. Scenarios provide easy-to-understand narrative descriptions of how users will interact with the system. Critical design decisions and tradeoffs are encapsulated in the claims, and allow developers to quickly compare different design options. It also allows them to justify design decisions to other stakeholders and to better plan for and direct meetings with clients or users.

4.2.3 Usability evaluations through the CDR

The other addition we made to the existing XP framework is usability evaluations. XP includes unit testing which verifies the functional accuracy of the code, and acceptance testing which proves to customers that the system works as agreed upon. Usability testing will verify that the system
is easy and intuitive to end users and is critical to closing the feedback loop that will support continual improvement of the user interface. Although there may be some overlap between acceptance and usability testing, the focus of each is distinct and equally important. Scenario-based design, the CDR, and related techniques from usability engineering provide the framework and guidance necessary to support usability evaluations.

Claims in the CDR are used to analyze and reason about specific interface features. Claim downsides or upsides can be validated through usability evaluations and help developers identify usability problems. By tracking which claims correspond to the stories currently being developed and looking at their interrelationships in the claims maps, developers can plan targeted evaluations at the end of each iteration. They also leverage light-weight usability evaluation methods such as expert walkthroughs and heuristic evaluations to quickly evaluate the interface. This process complements the XP practice of test-driven development to maintain functional correctness of the code. In this case, 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.

4.2.4 Design case studies

Two design case studies were conducted to demonstrate how the XSBD process addresses the key questions detailed in chapter 1. The developers were four undergraduate students receiving research or independent study credit. Three people, including the authors of this paper, acted as managers and oversaw each development project. The students were introduced to the XSBD process over the course of several weeks at the beginning of the semester. The remaining 10 weeks were devoted to development.

Each group used the XSBD process to develop their respective systems. Development proceeded over the course of five two week iterations, representing a single release cycle. Project information and CDR documentation was stored on a wiki that the developers and clients could access at any points in the project [43]. The development environment was made as realistic as possible but there were several limitations. First, the developers and clients were all located at the Blacksburg Campus at Virginia Tech but developers and clients were not collocated in a single office environment due to work and academic obligations. However, developers at least held weekly meetings with their clients and remained in constant email contact. In addition, pair programming was not used consistently throughout the semester due to conflicting schedules. They were advised to conduct code reviews together when they had to work separately. Project managers did a code walkthrough with one or the other member of a team (including unit tests) at the end of each iteration to verify that both
developers in each team understood the code. Detailed results from the case studies can be found in the Agile 2007 submission [41].

Evolution of the CDR

Figure 9. Example claims map. The root concept claim (green) is linked to activity claims (blue) which are linked to implementation claims (yellow). Note the progressive growth through the iterations. (Note that only claim features are included in the diagrams)

The CDR was used to maintain a coherent, consistent and understandable interaction architecture within the incremental agile development process. It provided a broad overview of the design and specific details when needed to guide usability evaluations. The relationship between the
CDR and the stories being developed enabled the developers to make key decisions in the development process. Each project began with a release planning meeting involving the developers and their respective clients. System conceptualization, stakeholder identification, and high-level design goal formation were done in this initial planning meeting and iteration.

After developing an initial list of design scenarios, specific claims are developed that correspond to the tasks that each system needs to support. The claims are arranged into a claims map, which shows how the different tasks and interface features are related to one another. The claims map starts with a root concept claim, which describes the system being developed in addition to important tradeoffs that need to be considered. These tradeoffs can relate to a mix of technological, contextual and usability issues.

Activity claims radiate out from the root concept claim and correspond to the specific tasks the system should enable. More specific claims are then linked to the activity claims which describe exactly how those tasks are supported. These specific claims can describe how information is displayed to the user and what interactions are supported. They generally correspond directly to one of the stories. In the first iteration, the claims map primarily consists of the root concept claim and the activity claims. At each iteration, developers implement a small subset of the total functionality of the system. Similarly, the CDR grows incrementally as this functionality is developed (Figure 9). The organization of the CDR shows the design of the system at multiple specificities—from the conceptual level to the task level to the interface level. Maintaining this organization gives developers a constant overview of the most important features of the interface, the tasks they support, and how they are interrelated. This claims map allows developers to refactor the interface to reflect new usage scenarios and changes in response to client feedback and evaluations. A detailed discussion of the relationships used in the claims maps is available here [78]. The claims map becomes increasingly complex as design proceeds, but the root concept claim and activity claims are relatively stable throughout the development process although they can change from iteration to iteration.

**Connecting SBD and XP**

In the XP+SBD process, usability engineering and system development occurs in a single unified development process. In larger teams, different developers may focus more on usability or development depending on their expertise, similar to the development process followed by Lynn Miller and the team at Alias [52]. The important point is that both software development and usability issues are considered concurrently throughout the development process. This ensures that the entire team understands both aspects of development and how they interact. One group demonstrated how different concerns can overlap when they were forced to redesign their interface to deal with a limitation of the location-detecting technology they were using.
Evaluating the Interface

Evaluating the usability of the interface, unlike unit testing, is difficult to completely automate. Evaluations can result in a large amount of quantitative and qualitative data that touch on many different aspects of usability including ease of use, learnability and overall satisfaction. They also require an actual person or persons to conduct the testing. As a result, the XP+SBD process advocates the use of light-weight analytic evaluations, which occur at the end of each iteration followed by more in-depth usability evaluations at the end of each release cycle. The CDR is used to guide these evaluations by helping developers determine what areas of the interface—described in claims—to evaluate at the end of each iteration and what effect redesigning some part of the interface will have on other parts of the system. The only time where a large amount of time was spent running an evaluation was at the end of the semester when a comprehensive usability evaluation of the entire system was conducted. This is needed at the end of release cycles to validate overall usability and uncover additional usability problems.

Communicating Design Rationale

The different parts of the CDR facilitated communication of design rationale among project stakeholders. The student developers were not explicitly told what parts of the CDR to share with their clients but they were encouraged to share materials they thought could facilitate communication. High level design goals and scenarios proved to be useful for conveying information to clients. Claims were not explicitly shared with clients. Developers relied more on verbal communication of design concerns and issues.

4.2.5 Discussion of results

Table 2 summarizes how the XSBD approach addresses the tensions between agile methods and usability. The incremental development illustrated by the CDR showed how it was possible to develop a coherent and consistent interface design representation by continuously reviewing the claims map and refactoring the interface when the need arose. This allowed agile developers to maintain a consistent, incremental release cycle throughout the project. However, it can be difficult to determine how complete a design representation should be. The developers in both projects largely relied on their own judgment as to what parts of the interface should be represented and which did not need to be. There is a tradeoff between completeness and manageability of the representation that developers will have to balance when using a design representation like the CDR. Critical parameters were shown to be a useful guide for defining overall project goals and measuring progress. However critical parameters can be difficult to define or measure depending on the type of system being
developed. The developers relied mostly on rough estimates for the IRC values which still proved useful in communicating design goals among themselves and their clients.

The developers use of claims and claims map showed how useful it can be to maintain a list of interface design decisions and the potential tradeoffs they entail. These helped them to see exactly what parts of the interface needed to be tested in the current iteration and which could be deferred or ignored. Many of the evaluations were in fact folded into regular client meetings.

Table 2. Table shows how the XSBD approach addresses tensions between agile methods and usability.

<table>
<thead>
<tr>
<th>How can developers design consistent and coherent interface architectures within an incremental agile development framework?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental development of an interface supported by a design representation like the CDR</td>
</tr>
<tr>
<td>+ Can help developers maintain consistent and cohesive interaction design through the continual evaluation of design rationale and targeted interface improvements.</td>
</tr>
<tr>
<td>+ Does not limit or excessively delay incremental software delivery</td>
</tr>
<tr>
<td>- Can be difficult to determine when a design representation is sufficiently complete</td>
</tr>
<tr>
<td>Using critical parameters like IRC values to guide interface development</td>
</tr>
<tr>
<td>+ Can be used to measure design success through repeated evaluation of explicit metrics</td>
</tr>
<tr>
<td>- Critical parameters may be difficult to define and measure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How can usability evaluations be streamlined so they better fit in accelerated development cycles while still providing useful results?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintaining an organized list of design tradeoffs, to guide lightweight usability evaluations</td>
</tr>
<tr>
<td>+ Can allow designers to target specific areas of the interface to evaluate, thereby saving effort and reducing the need to reevaluate parts of the interface in later iterations</td>
</tr>
<tr>
<td>- Requires additional effort by developers to plan and run usability evaluations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How can project members support communication and cooperation between designers, customers, users and other stakeholders who have different backgrounds and expertise?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared design representation showing high-level and detailed views of the interaction design</td>
</tr>
<tr>
<td>+ Allow different stakeholder groups with different backgrounds to understand and give feedback about the design.</td>
</tr>
<tr>
<td>+ Makes interplay between usability and agile development work explicit and understandable</td>
</tr>
<tr>
<td>+ Can focus planning by reminding stakeholders of key design decisions and concerns</td>
</tr>
<tr>
<td>- Requires designers to actively maintain links between agile and usability artifacts</td>
</tr>
</tbody>
</table>

The cases also show that a design representation consisting of easy to understand artifacts that make explicit the connections between the different concerns of the stakeholder groups can support
communication and cooperation. Communication and buy-in is vital for these different groups to work together effectively. This representation allowed different stakeholders to have a shared understanding of the overall design and to make informed tradeoffs when conflicts came up. However, maintaining this kind of representation does take some effort.

4.2.6 Summary of study findings

To that end, this work has focused on finding ways for agile software developers and usability engineers to work together more effectively by addressing the conflicts between extreme programming and scenario-based design. We end with four guidelines for usability specialists and agile practitioners that can be derived from this work.

- Share design documents and artifacts when possible. Maintaining communication among team members is vital. Sharing these artifacts can augment face-to-face communications and allow developers to make informed decisions about design tradeoffs.

- Strive for continuous interface improvement. Incremental additions to an interface can gradually erode overall usability. Always be aware of possible improvements and do not be afraid to test new ideas. A larger number of smaller, more focused usability studies can result in a similar (or better) level of understanding as a small number of large tests—and it better fits with the agile philosophy.

- Integrate usability into day to day development tasks. Continuous improvements require continuous user feedback. Conduct informal evaluations when more complete usability evaluations are not feasible. Have clients or other team members look over new interface features. Such data can be valuable when you know who you’re designing for, what data you’re collecting and why you’re collecting it.

- Avoid having team members overspecialize in one area. Team cohesiveness is important to maintain velocity. Members with separate focus areas/expertise should have an understanding of each other’s specialties to prevent misunderstandings and wasted work.

4.3 Chapter summary

This chapter summarized the first stage of my dissertation focus—developing ways to integrate usability practices and tools into existing agile practices without violating the key tenets of agile software development. Specifically, it describes my work in integrating scenario-based development
and extreme programming through the central design record. It focused on how I first developed the
CDR to help novice developers learn and apply usability concepts while developing a system. I then
adapted the CDR as a way to not only capture the interface design, but to also aid in collaboration and
communication between usability professionals and software developers.

This work shows how the CDR can help developers to maintain a coherent interface and
interaction architecture within an agile development environment by allowing usability engineers to
incrementally develop and improve the interface while maintaining a high-level overview of the
design. However, there are still a number of issues that need to be resolved before the XSBD process
and the CDR can be used in practice. First, it was shown that different aspects of the CDR such as
claims are not easily shared with different stakeholders. This indicates that the CDR as it is currently
constructed may need to be changed to better support communication between different stakeholders.
Second, results show that it is sometimes difficult to manage the CDR while trying to maintain a tight
development schedule. There is a need to streamline and adapt aspects of the CDR to reflect the needs
and concerns of different stakeholders so development decisions and general project management can
proceed more smoothly. These concerns will be addressed in the following proposed work.


5 Proposed work: bringing XSBD to the real world

The completed work introduced the XSBD process, a usability-centric approach to developing usable systems within an agile framework. Previous studies showed that the central design record can be used by usability engineers to maintain a consistent and coherent interaction design while being incrementally developed [38][41]. It was also shown that maintaining an organized list of design tradeoffs, like a claims map, can help designers target specific areas of the interface to evaluate thereby saving effort and reducing the need to reevaluate parts of the interface in later iterations [39][40][42]. This is important in an agile process since usability testing is not easily automated. Finally, it was shown that a shared design representation such as the CDR may help usability engineers and other project stakeholders to communicate and collaborate more effectively [40][41]. The CDR thus represents a solution to the third research question presented in Chapter 1—How can designers develop & maintain a coherent interface and interaction architecture design in an agile environment.

However, several challenges remain in determining how to better integrate the CDR and its associated practices into an agile development team so that the team can work together effectively and so that project velocity is maintained throughout the life of the design project. Feedback from agile practitioners have criticized the development and use of design representations as too heavyweight and have questioned how exactly claims will be used to plan and run usability evaluations effectively. In addition, previous studies have not validated that separate usability and software development teams can communicate effectively through the CDR [39][40][42]. Those studies relied on student developers to act as both software engineers and usability engineers. The proposed work will answer the remaining two research issues. First, this work will clearly define how project velocity can be maintained in an agile project with the addition of usability practices and tools. Second, this work will more completely answer the first research question—how to converge on the satisfaction of requirements from different stakeholders—by developing ways that usability engineers can present and filter information from the CDR to more effectively communicate with software developers. One of the key driving metrics used to determine whether a practice belongs in an agile process is to ask whether it brings one closer to a functioning work product [68]. This proposed work will provide answers to the first two research issues introduced in the first chapter and address the concerns of agile professionals. The key driving metric of this work will be to ask how a particular practice brings one closer to a work product that satisfies usability as well as functional requirements.
5.1 Improving communication and collaboration through the CDR

In the XSBD approach, the CDR is presented as the common point through which usability engineers and software developers can reason about and make informed design decisions that take into account the needs and concerns of both groups. However, certain aspects of the CDR that are important to usability engineers appear to be too complex and separate from development concerns to be of much use to software developers. Similarly, certain artifacts from the software side of development such as the code or the individual unit tests that validate functions are not particularly meaningful or useful for usability specialists. There is a need to determine what parts of the CDR are useful for other stakeholders. In addition, there is a concern that the CDR itself, particularly the claims map, is too complex and time-consuming to develop [39][42]. There is need to more explicitly define what the purpose of each part of the CDR is and identify ways to present the CDR to other stakeholders so it is easier to understand. These issues are summarized below:

- What aspects of a design rationale-based design representation, like the CDR, need to be shared and shown to agile practitioners for them to be able to understand usability concerns in the context of the overall design and make meaningful design decisions?
- How can and should design rationale-based design representations be modified and presented to other stakeholders so that it is easier for them to understand and use?

5.1.1 Defining and streamlining the CDR for better communication

The discussion below represents my preliminary approach to addressing the above issues. First, I will define what the responsibilities for each stakeholder are in terms of developing and maintaining the CDR and will define specific parts of the central design record that will be shared with non-usability stakeholders. Then, I will identify ways to present the CDR to other stakeholders so they both understand the design and the underlying design decisions embedded within them.

Usability engineers will be tasked with developing high level usability design goals, user profiles, and design the user interface through iterative cycles of prototype development and user evaluations. As such, they will be primarily responsible for developing and maintaining the CDR and other supporting artifacts. Developers need some understanding of usability-related information such as overall design goals, users roles, and what tasks need to be supported. However, there is typically no need for them to see things such as detailed usability evaluation data, or even claim tradeoffs for different aspects of the interface. Similarly, usability engineers may need to understand at a high level what implementation limitations exist, but they do not need to go through the actual code or
algorithms used to implement functionality. To promote effective communication, all stakeholders need a shallow understanding across their respective areas of expertise so they can collaborate more effectively and make informed design tradeoffs effectively. Previous work has defined the CDR and how it is used by usability engineers, but has not defined how information from the CDR will be filtered and presented to other stakeholders.

![Diagram](image)

**Figure 10.** Usability engineers present to other stakeholders only the parts of the CDR that help them to easily understand and reason about the design.

A second concern is that the CDR, particularly the claims map which developers use to actively maintain relationships between design features, may be seen as too complex and too far removed from what agile developers and other stakeholders care about in terms of the interface design. The CDR is meant to support, not replace, existing prototyping methods such as paper sketches, or low-fidelity prototypes made in applications such as PowerPoint or Visio. The value of CDR artifacts such as claims and claims maps will be in identifying critical interface design tradeoffs and in planning usability evaluations. One solution may be to present design prototypes as the primary medium through which to communicate the design to other stakeholders (Figure 10). This will include the use of scenarios, which are easily understood and useful for providing context [14][71]. Another problem may be in the way that CDR artifacts are presented to other members of the agile team. For example, previous studies used electronic tools to develop and maintain CDR artifacts such as the claims map [41][42]. Many agile teams rely on physical artifacts such as note cards when identifying and prioritizing design features and when tracking project progress [1][52][65]. A simplified physical representation of the CDR could be used to track and present usability information to other
stakeholders. For example, note cards could be arranged on a wall to represent the claims map for the current design with colored post-it notes used to identify things such as what parts of the system are currently being implemented or which features have been tested. More detailed information such as complete claims, usability evaluation plans and evaluation results could be managed separately by usability specialists. Of core importance will be presenting the usability information captured in the CDR that best allows the entire team to work towards a functional system that meets high level usability design goals.

5.1.2 Success measures

New views of the CDR will be developed specifically for developers and other stakeholders including clients. This will not involve the development of additional design representations but rather the selective presentation of elements of the CDR. In addition, I will develop different ways of presenting usability information from the CDR such as the post-it note representation referenced above and look at when and how such representations should be used in the design process. The success of this modified representation will rely primarily on subjective data from project stakeholders—specifically how well they understand high level design goals, interaction architecture, and relevant usability issues uncovered in evaluations.

5.2 Maintaining project velocity

In XP and other agile approaches, development artifacts such as story cards are used to rank and schedule development tasks. By measuring the number and rate at which story cards are implemented, developers can get a sense of a project’s velocity. Although the connection between stories and claims has been defined in the XSBD process, there is still a need for how these project management techniques will account for the completion of usability tasks. Scheduling usability tasks in terms of agile project management practices will also help designers to make more informed design tradeoffs by getting a sense of the severity of a usability problem in addition what it would cost to resolve the issue in terms of development effort. A related problem is how exactly to measure project velocity in terms of high level usability goals. This can be measured using a number of different metrics including how closely a design is aligned with its critical parameters goals and the number of claims that have been validated. Project members will then be able to better gauge how closely they are approaching high level usability design goals. These issues are summarized below:

- How to define and schedule usability tasks to be compatible with existing project management practices in XP.
• Identifying how usability metrics can be derived so developers can measure project velocity in terms of usability.

5.2.1 Scheduling usability in an agile project

In XP projects, each person estimates work based on the length of an iteration. For example, if an iteration lasts two weeks, then each person in the project can work for 80 hours (assuming a five day work week and an eight hour work day). Each developer can then be assigned tasks based on the highest priority stories that need to be implemented. This allows managers both to estimate what parts of the system will be complete at the end of each iteration and allows them to make adjustments to the schedule during an iteration based on the rate of completion for those tasks. For example, if a manager notices tasks are being completed at a slower rate than expected, he can make adjustments by removing tasks from the iteration. Usability task scheduling should be similarly scheduled so that managers are able to measure progress of usability task completion and make more informed decisions about what tasks should be done in the next iteration.

Figure 11. Cards can be used to present usability problems to other project stakeholders. They can then be used to collaboratively work out what issues can and should be resolved.

Users did not understand meaning of icons in control menu. Often took several seconds to find and select the appropriate control.

Severity: High
Evidence: Strong
Redesign Effort: Low

The proposed solution is to define usability tasks using the same basic techniques as agile process like XP or SCRUM [4][69]. That is, if usability specialists work for five days a week and work eight hours a day and each iteration is two weeks, then each usability specialist has 80 hours of work to schedule in an iteration.. Usability tasks will consist of things such as running usability evaluations, analyzing usability results, and prototyping. These tasks will be prioritized based on what things need to be done, the schedule and on a number of usability metrics. Project managers will then be able to gauge project progress in terms of functional completeness and overall usability. For example, project management decisions about whether or not to resolve a usability problem can then be made based on the following factors:
• What is the severity of the usability problem? (i.e. How closely does it relate to high level design goals?)
• What evidence is there that the problem exists? (i.e. what type of test was run and with who?)
• How much effort will be required to resolve the problem (in both design and implementation terms?)

A key part of the usability engineer’s job will be to present work in ways that will allow the team to work together towards the common goal of delivering a working, usable product. This will involve summarizing, simplifying and organizing usability data, derived from the CDR and usability evaluations, in ways that are easier for non-usability specialists to understand (Figure 11). It will also involve working more closely with developers to understand implementation-related implications of proposed changes. Work needs to be done to develop how this information will be presented and how it will be used in planning and evaluation meetings in the XSBD process.

5.2.2 Deriving usability metrics to measure usability velocity

In agile methods like XP, it is important to be able to measure the rate at which development tasks are being completed and where bottlenecks are occurring so adjustments can be made to maintain project velocity. To effectively integrate usability processes into an agile environment, it will be important to develop ways to measure the rate at which usability tasks are being completed and the quality with which they are completed. Development tasks have well-defined ways to determine completeness. Code can be verified automatically through the common unit test suite that emerges from the test driven development process. Similarly, functional correctness is verified by having customers check to see if requested functionality is included in the software and works correctly. Validation of some usability features and tests cannot be defined exactly. For example, a usability evaluation may show that a usability problem may or may not exist. Or the estimated severity of the problem may vary depending on the type of usability evaluation that was run.

The proposed solution will build on the use of critical parameters by having usability engineers provide not only an estimate of how closely a design is approaching usability goals, but also provide an estimate of confidence based on what evidence is collected and how it was collected. This will allow the design team to better gauge the validity of the interface design and will help in making project management decisions as mentioned in the previous section. In addition, to promote transparency among developers and usability professionals on the team, usability work will be tracked using metrics similar to those used to track development velocity. Team members will be able to see roughly what other members are working on and will allow managers to identify problems that could
slow the development process. For example, a simple metric could be to measure the number of
claims tested and validated in a given iteration, or the number and severity of usability problems
uncovered in a usability test.

5.2.3 Success measures

This work will result in a well-defined process for scheduling and prioritizing usability tasks
within the agile framework. To support this, I will also develop a way to structure and present
usability results to other project stakeholders so project managers can make more informed design
decisions. Subjective feedback from project stakeholders will allow me to compare the relative
success of this approach to previous approaches in terms of scheduling usability tasks and measuring
usability progress. In addition, development progress using this approach will be compared against
previous agile development projects using standard metrics such as the number of work items
completed in an iteration or the number of bugs that are resolved in an iteration, to see if project
velocity can be maintained even with the addition of usability practices.

5.3 Developing a comprehensive solution to agile usability

This chapter summarized my proposed work to further develop the extreme scenario-based
development process. My previous work has shown that the XSBD process, with the CDR as central
to the process, can be used by usability engineers to incrementally develop a coherent interaction
design through targeted usability evaluations with end users. However, two key open questions still
need to be resolved for the process to be usable within a practical development environment. First,
how exactly can usability engineers use the CDR to communicate with other stakeholders? Second,
how can usability engineers work in an agile team to maintain project velocity while effectively taking
into account usability within the context of other development concerns?

In answering these questions, the guiding metric of this proposed work is to determine
whether a particular practice will lead to a usable and functional end product. The general approach
was to find ways of filtering and presenting information in the CDR to developers and project
managers so they only see the information they need to communicate with each other and make design
and scheduling decisions. Figure 12 shows this general filter approach. Similar to how usability
evaluations present end users with specific information tailored to help them evaluate the interface,
information from the CDR will be filtered to support project management and system development.
This approach will make it possible to use the XSBD approach in a practical development
environment. The research case study that will validate this proposed work is presented in the next
chapter. The timeline and listing of key deliverables of this proposed work is presented in chapter 7.
Figure 12. Only relevant usability information is presented to different stakeholders to support project management, development, and acceptance testing.
6 Proposed research case study

To effectively study the proposed additions to the XSBD process, I will use a case study approach as well as periodic development process retrospectives. Case studies are used increasing to conduct research in software engineering and to evaluate new methodologies and tools both because of their flexibility and because context is important and difficult to simulate in more controlled settings [67][82]. Rather then look at small portions of the process in a controlled setting, I will analyze the process in a broader, more realistic development environment though I will focus data collection and analysis activities on answering specific questions. Elements of our approach have been pilot tested in small student-led and classroom-based development projects and have produced some promising results [39][40][41][42]. However, as noted previously, one problem with these studies, and common criticism of this approach from industry-oriented publications, is that these studies provide insufficient evidence that the agile usability practices and design representations have utility in an industry setting. By partnering with a software development firm and guided by my prior research and theories in XSBD, I will present results that show how the proposed additions to the approach facilitate communication among different stakeholder groups within an agile team, and how agile teams can maintain project velocity even with the addition of usability engineering processes.

This chapter will detail a preliminary plan for the case study, which will be run at Meridium offices in Roanoke starting in early 2008.

6.1 Motivation

The proposed additions to the XSBD process focus on improving team coordination and communication in an agile process. A case study in an actual development environment will provide practical data about how the approach works in practice and will allow me to better study the purported benefits of the additions; namely improved communication among project stakeholders and the ability to maintain project velocity through better integration of usability practices into the project planning process.

This approach will include a more experienced and diverse agile team that will include practicing developers including a project manager, software engineers, a usability engineer and a quality assurance specialist. This will allow me to collect data about how different stakeholders, especially software developers and usability engineers, can communicate and coordinate with each other using the XSBD process. Also, unlike previous studies which included student developers who had classes and other obligations, the team members in this study will be able dedicate a more consistent amount of time to the project. This will allow me to better gauge project velocity and to
compare the velocity of this project with similar past development efforts. Finally, agile practitioners and researchers have acknowledged the need to find ways to integrate usability into an agile process but have questioned the utility of the XSBD approach because it has up to this point only been evaluated in a controlled, academic environment [39][40][41][42]. This approach will partly address these concerns by providing a case study of the XSBD process in an actual end-to-end development effort.

6.2 Method

For this study, I will be partnering with Meridium, Inc., a Roanoke-based software firm that focuses on developing asset strategy management software and services for gas, coal, oil and other large industrial companies [51]. Meridium is currently working to transition to a more agile development process and I have been working with them since January 2007 as a usability consultant and software developer. D. Scott McCrickard and I have submitted a joint VT-Meridium NSF Phase I Small Business Technology Transfer (STTR) proposal with key Meridium leadership to fund the case study that will evaluate the approach [55]. STTR Proposals are intended to encourage small businesses work with research institutions to develop novel intellectual property and tools. This 1-year effort will evaluate the new proposed XSBD approach and drive requirements for a larger scale 2-year phase II effort which will include the development of tools to support the approach. In order to control the scope of the project, this case study will focus on a single development team within Meridium who will follow and represent the general development practices at Meridium as an organization.

These development efforts will focus on mobile handheld application development, a new area of business that Meridium has begun to move into. These applications will be used to collect data on and analyze information related to physical equipment in industrial and factory settings. Mobile applications are ideal in that their success is heavily dependent on contextual and user interface issues that are often not analogous to standard desktop systems. In addition, I will be able to leverage my previous work in using scenario-based development and agile methods to develop handheld applications and notification systems [8][48][54][72][79].

The remainder of this section will describe the general approach for the case study and project retrospectives.

6.2.1 Case study approach

Before the case study can begin, I will introduce key aspects of the XSBD approach, the design representations used, and how they are used by software developers and usability specialists in
developing the system. This stage will also require myself and the participants at Meridium to specify and develop an understanding of existing development practices at Meridium, its overall organization, and how team members communicate with each other and with customers. Specifically, we will focus on how developers at Meridium will adapt and adopt the agile SBD processes and models, what organizational changes need to be made to Meridium’s existing development structure, and how best to integrate the process into the organization such that it accounts for factors such as scheduling, budget issues, and marketing issues.

The purpose of the case study will be to verify whether the proposed changes to the XSBD process answer the questions presented in Chapter 5. They are summarized below:

- What aspects of the CDR need to be shared and shown to agile practitioners for them to be able to understand usability concerns in the context of the overall design and make meaningful design decisions?
- How can and should the CDR be modified and presented to other stakeholders so that it is easier for them to understand and use?
- How should usability tasks to be defined and presented so they can support project management practices in XP?
- What usability metrics can help project managers to measure project velocity in terms of usability?

Since it will be difficult to differentiate between the effects of the process and specific contextual effects, data will be collected using a number of different methods. These different sources of data will need to converge in a triangulating fashion to be compared against the initial questions. First, project team members will complete questionnaires at a number of checkpoints throughout each development project. These questionnaires, which will include a mix of Likert-scale questions and written responses, are intended to gather data on how different parts of the CDR are used throughout the project, how team members are using the CDR to interact and communicate with each other, and how the use of the CDR and the agile SBD process in general is helping or hindering their progress. Results from these questionnaires will be analyzed using appropriate statistical methods to gain a quantitative understanding of how useful the approach is in general, and compare the perceived utility across different team member groups including developers, usability specialists, business analysts and customers. In addition to these subjective responses, the researchers will make copies of and analyze the work products from each project, particularly the CDR. They will look at things such as the relative size of each as the project progresses, the number of times specific claims and scenarios are changed, and how teams used them to guide periodic usability evaluations and measure design progress. This data will be compared against the qualitative results collected from the questionnaires.
to see if there is agreement in the results derived from each. In addition, I will collect standard development metrics collected by project managers and quality assurance specialists throughout the course of the project. These standard software metrics include things such as the number of open development tasks, the number of bugs, and how often the development schedule slips. These metrics will allow me to measure the project velocity and compare it against past agile development efforts at Meridium.

6.2.2 Project retrospectives

This design effort will also make use of project retrospectives [35]. Project retrospectives are used in software projects to reflect on and improve design practices. These retrospectives, conducted on a bi-weekly basis over the course of the project, are intended to give project participants an opportunity to learn about the XSBD processes and models in use and continuously improve them over the course of the project. Retrospectives are conducted with the entire team present and a facilitator, and generally involve looking at what did and did not work over the past iteration, what the underlying causes are, and what will be done to address problems and maintain successes. They help to create a sense of community and understanding across the entire team. Customer involvement will be key in these retrospectives to discuss successes of the approach and demonstrate overall team commitment to continuous improvement. These retrospectives will be used to gather subjective feedback on the XSBD approach and to make adjustments or changes if aspects of the approach are not working effectively.

6.3 Expected results

Based on the subjective responses of the participants, I expect that the developer participants at Meridium will agree that they are better able to make informed usability design decisions due to the use of evidence-based rationale in the CDR. I do not expect to be able to definitively say that using the approach will lead to systems with better usability. This is highly dependent on the experience and skills of the usability engineers and interaction designers on a particular project. Rather, I expect to be able to say that the agile development team will be able to make more informed design tradeoffs that take into account both usability and other factors including scheduling and budget. Customized views of the CDR and other usability results and artifacts will lead to better collaboration and communication among usability engineers and other stakeholders. I also expect to see, based on subjective responses from participants and by looking at past projects, that project managers are able to maintain project velocity while taking usability factors into account. Based on the project retrospectives and on summative questionnaires, I expect to see increasingly positive feedback on the
overall approach and expect participants to be more informed about the impact of usability within the context of the overall design and to note improved communication of usability issues by using the CDR in the XSBD approach.

There is a possibility that the overall subjective experience will not be significantly better than their existing agile approach. This is because the existing Meridium development process does not include established usability practices and the benefits of including usability may not be immediately obvious to the participants. However, I expect to receive valuable feedback that can be used to further improve the approach while better taking into account practical development concerns. In addition, the results will drive a subsequent 2-year Phase II STTR proposal which will focus on further improvements to the process and the development of tools to support the XSBD process and CDR management.
7 Summary of proposed work

This chapter will revisit the high-level goals of this work and look at its broad implications within the context of the software development industry. It will summarize key contributions and how they address the proposed research issues. It presents a timeline and a brief overview of the key milestones and deliverables leading to the completion of the proposed work.

7.1 Contributions and Impact

The emergence of agile software development practices represents an additional line of attack in addressing the increasing complexity and difficulty in developing software systems that face time and budget limitations and ever-changing requirements [4][29][32][69]. While agile methodologies have been accepted as an effective way to efficiently meet the technical and business needs of software projects, agile practitioners initially did not develop ways to incorporate usability practices into agile processes. As a result, it was possible for a system that was developed using an agile process to meet functional and business objectives but have poor usability [65]. Agile practitioners and acknowledged this problem and have actively been developing ways to integrate usability into agile practices [7][20][24][52][65][68][74].

My goal is to show that the results from the proposed case study and my past work verify that the central design record and the associated practices in this XSBD approach provides a way to integrate agile development processes and usability engineering processes through a common design representation that improves team communication and allows for continuous interaction design improvement through targeted, light-weight usability evaluations. Specifically, it will address the three key research issues introduced in chapter 1 as follows:

1) Custom views of usability results developed by usability engineers and derived from the central design record will help stakeholders with differing motivations, needs and concerns to communicate more effectively with each other and to make informed decisions that account for usability factors.

2) Scheduling and defining usability tasks in concert with existing agile project management practices combined with clearly defined critical parameters will allow agile teams to maintain project velocity through a common task prioritization process guided by a unified set of high level design goals.

3) Agile teams can incrementally develop a user interface in an agile project while maintaining a coherent interaction architecture design through the consistent development, evaluation and refactoring of the interface through the central design record.
This approach describes a comprehensive ‘one team’ approach where agile practitioners and usability specialists work together as part of one team. By exploring how two specific design processes from both areas (extreme programming and scenario-based design) can be integrated, this work will provide actionable guidance for practitioners and will result in generalized guidelines—many of which are expected to validate and reinforce the work of other researchers in the field. In addition, this work serves as a way to address the theory-practice gap between academic HCI research and industry [13][15][17][37][70][81]. I hope to show one way that theory-based usability work developed in academia can be applied in practice and that industry knowledge can be brought to bear in improving and building on those theories. Finally, the CDR artifacts that are developed in agile projects could serve as a record of design decisions that can be recorded for future use—possibly as a part of educational case studies or as a reusable knowledge repository that could be applied to future agile projects.

### 7.2 Timeline and key deliverables

A timeline of the proposed work is shown in Figure 13. ‘Approach Development’ refers to the additions and changes that will be made to the XSBD process to address the challenges as described in chapter 5. This overlaps with the running of the case study as improvements and changes to the approach will be made based on how well it works in practice. ‘Plan Study’ refers to the development of an initial detailed plan for the development case study at Meridium. This overlaps with the approach development as certain details of the approach will be developed based on feedback from developers at Meridium. The ‘Case Study’ will cover approximately six months although this may change somewhat depending on the scope of the system developed using the methods described in Chapter 6. Preliminary analysis of results will begin at the end of the first iteration and will continue throughout the length of the case study although detailed analysis of results will occur after the final project retrospective at the end of the study.
The key peer-reviewed deliverables are indicated by the inverted triangles in Figure 13. They are detailed below:

(A) Workshop Submission to Agile 2008 Developers Conference [http://www.agile2008.org/]. This workshop submission will build on my previous research paper on the XSBD process which was presented at the Agile 2007 Developers Conference and will incorporate aspects from the proposed work. This submission will better introduce the details of the approach to agile practitioners by giving them a hands-on demonstration of how it works. It will also provide more informed feedback about the effectiveness of the approach.

(B) Journal Submission to International Journal of Usability Studies [http://www.upassoc.org/upa_publications/jus/index.html]. It will be important to introduce this approach not only to agile practitioners, but also to usability professionals who could potentially members of agile teams. This submission is intended to get feedback from and to disseminate information about my approach to the professional usability community.

(C) Submission to the Agile Journal [http://www.agilejournal.com/]. This is an online journal with a readership of over 20,000—primarily agile practitioners. This submission is intended to introduce the XSBD approach to the agile community at large. This exposure could
potentially drive further discussions of my approach at future workshops and conferences and lead to future collaborations and employment opportunities.

(D) NSF STTR Phase II Proposal [http://www.nsf.gov/eng/iip/sbir/]. This two year grant will be a direct continuation of this dissertation work and will focus on the broader application of this approach within the software industry and the development of software tools to support it.
References

29. Fowler, M. Is design dead? (2002). In Auer, K., and Miller, R. Extreme programming examined, Addison-Wesley, Boston, MA


64. Olsen, G. (2005). The emperor has no lab coat, Interactions, 9(4), 13-17


