Connecting the usability and software engineering life cycles through a communication-fostering software development framework and cross-pollinated computer science courses

By

Pardha S. Pyla

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Computer Science and Applications

Research directed by: Dr. H. Rex Hartson

Committee: Dr. H. Rex Hartson, Dr. James D. Arthur, Dr. Tonya L. Smith-Jackson, Dr. Manuel A. Pérez-Quiñones, and Dr. Deborah Hix.

September 10, 2007
Blacksburg, Virginia

Keywords: Software engineering, usability engineering, integration, unified framework, life cycle representation, influencing factors, Ripple.

Connecting the usability and software engineering life cycles through a communication-fostering software development framework and cross-pollinated computer science courses

By

Pardha S. Pyla

Abstract

Interactive software systems have both functional and user interface components. User interface design and development requires specialized usability engineering (UE) knowledge, training, and experience in topics such as psychology, cognition, specialized design guidelines, and task analysis. The design and development of a functional core requires specialized software engineering (SE) knowledge, training, and experience in topics such as algorithms, data structures, software architectures, calling structures, and database management.

Given that the user interface and the functional core are two closely coupled components of an interactive software system, with each constraining the design of the other, there is a need for the SE and UE life cycles to be connected to support communication among roles between the two development life cycles. Additionally, there is a corresponding need for appropriate computer science curricula to train the SE and UE roles about the connections between the two processes.

In this dissertation, we connected the SE and UE life cycles by creating the Ripple project development environment which fosters communication between the SE and UE roles and by creating a graduate-level cross-pollinated SE-UE joint course offering, with student teams spanning the two classes, to educate students about the intricacies of interactive-software development. Using this joint course we simulated different conditions of interactive-software development (i.e. with different types of project constraints and role playing) and assigned different teams to these conditions. As part of semester-long class projects these teams developed prototype systems for a real client using their assigned development condition. Two of the total of eight teams in this study used the Ripple framework.

As part of this experimental course offering, various instruments were employed throughout the semester to assess the effectiveness of a framework like Ripple and to investigate candidate factors that impact the quality of product and process of interactive-software systems. The study highlighted the importance of communication among the SE and UE roles and exemplified the need for the two roles to respect each other and to have the willingness to work with one another. Also, there appears to exist an inherent conflict of interest when the same people play both UE and SE roles as they seem to choose user interface features that are easy to implement and not necessarily easy to use by system's target users. Regarding pedagogy, students in this study indicated that this joint SE-UE course was more useful in learning about interactive-software development and that it provided a better learning experience than traditional SE-only or UE-only courses.

Don’t panic!

To

My mom, dad, and brother

*… for all the **love***

The 32 Hokies lost on April 16th 2007

*… for all the **promise***

Acknowledgements

I would like to acknowledge all the help, guidance, and support of my dissertation advisory committee:

- Dr. H. Rex Hartson, my advisor, for directing this research from conception through execution, for collaborating with me in running the evaluation components in this work, for teaching me the nuances of technical writing, for the countless hours he spent "debugging" my dissertation, and most importantly, for being a wonderful role model, mentor, guide, friend, philosopher, and a true inspiration in not only my research but life in general
- Dr. James D. Arthur for all the insightful debates on which is more important: software engineering or usability engineering? (he is yet to realize the truth that usability is, of course, more important), and for collaborating with me in running the evaluation components in this work
- Dr. Tonya L. Smith-Jackson for all her encouragement and extensive help with, and discussion about, experimental design and analysis
- Dr. Manuel A. Pérez-Quiñones for his unique insights into user interface software concepts, for his overall encouragement and guidance, and for supporting me through the past two semesters
- Dr. Deborah Hix for changing the direction of this dissertation (for the better) with one radical idea back in 2004

I would also like to acknowledge:

- My family for their unconditional love and encouragement, without which, this dissertation would not have been possible
- Dr. Scott McCrickard for his constant encouragement and for supporting me the past two summers
- Dr. Jonathan Howarth for being a friend, his help as the co-GTA of the joint CS5704/CS5714 offering in Fall 2006, and for making the past few years interesting with all the deeply philosophical discussions on usability of Microsoft products
- Dr. Cortney Martin for her help with statistical analysis in JMP and for introducing me to the joy of fishing for the ever elusive "statistical significance"
- Dr. Robert Capra for his encouragement and for all the interesting conversations in 3160
- Dr. Glenda Scales for having that one conversation with me which gave a whole different perspective on how to look at my data and for her encouragement
- Kamran Razvan from ClickAndPledge.com, for being kind enough to provide test accounts for credit card authentication for all the teams in my evaluation
- Rachel Smith, Rachel Horton, and Alec Selz from the Horticulture Club for their dedication and commitment as clients in my evaluation, and for showing me how cool horticulture is
- All the students in CS5704 and CS5714 who went out of their way to write detailed entries in their journals and for not complaining too much about the extra work they had to do as a result of the evaluation component in the class
- The staff at the Department of Computer Science, especially Dr. Calvin Ribbens, Ginger, Rachel, Melanie, Carol, and Tess, for their help in all things administrative
- All my friends, colleagues, and special ones who with their encouragement and support made my doctoral journey pleasant and fun. It is impossible to list all of them here, but some names that come to mind are Aparna, Beth, Boby, Catherine, Chris(s), David, Edgardo, Emilee, Jamika, Jean, Jian, Lauren, Laurian, Lee, Manas, Meg, Mehdi, Michelle, Miten, Monica, Nicholas, Nicole, Prachi, Ramesh, Ramya, Ranjit, Rhonda, Rieky, Sandeep, Sarah, Satish, Shahtab, Tej, Uma, Yonca, Young-Lan, ...

Table of Contents

List of Figures

List of Tables

1 Chapter One: Introduction

1.1 Context

Interactive software systems have both functional and user interface parts. Although the separation of code into two clearly identifiable components is not always possible, the two parts exist conceptually and each must be developed on its own terms with its own roles for developers. We use the term "developer" to refer to someone who has the skills to participate in any stage of either the functional or user-interface parts of system development, and not just on the functional software side.

The user-interface part, which often accounts for half or more of the total lines of code (Myers and Rosson, 1992), begins as an interaction design and is ultimately implemented in user interface software. Interaction design requires specialized usability engineering (UE) knowledge, training, and experience in topics such as human psychology, cognition, visual perception, specialized design guidelines, task analysis, etc. Traditionally, the concepts, theory and techniques associated with the user interface (UI) design domain are taught as a part of Human Computer Interaction (HCI) courses in most computer science departments. The goal of UE is to create systems with measurably high usability, i.e., systems that are easy to learn, easy to use, and satisfying to their users. A practical objective is also to provide interaction design that can be used to build the user interface component of a system by software engineers. We define the UE role as that of the developer (sometimes called the interaction developer) who has responsibility for creating such interaction designs.

The functional part of a software system, sometimes called the functional core, is manifest as the non-user-interface software. The design and development of this functional core requires specialized software engineering (SE) knowledge, training, and experience in topics such as algorithms, data structures, software architectures, calling structures, database management, etc. Traditionally, the concepts, theory and techniques associated with the functional core design domain are taught as a part of Software Engineering and other courses in most computer science departments. The goal of SE is

to create efficient and reliable software systems containing the specified functionality, as well as integrating and implementing the interactive portion of the system. We define the SE role as that of the developer (sometimes called the software developer) who has the responsibility for this goal.

To achieve the UE and SE goals for an interactive system, i.e., to create an efficient and reliable system with required functionality and high usability, effective development processes are required for both the UE and SE life cycles. The UE development life cycle (Figure 1) is an iteration of activities for requirement analysis (e.g., needs, task, work flow, user class analysis), interaction design (e.g., usage scenarios, screen designs), prototype development, and evaluation thereby producing a user interface interaction design specification. (Often, the specification *is* the design, in the form of an iterated and refined prototype.)

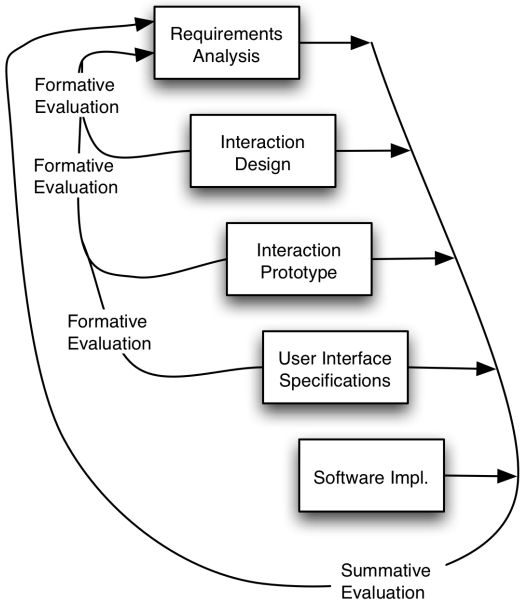


Figure 1: Usability engineering life cycle

The SE development life cycle (Figure 2) consists primarily of concept definition and requirements engineering, design (preliminary and detailed design), design review, implementation, and integration & testing (I&T).

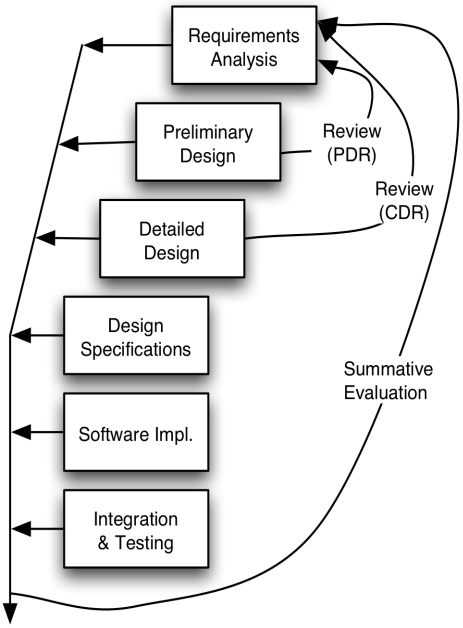


Figure 2: Software engineering life cycle

1.2 Problem Statement

Given that the UI and the functional core are two closely coupled components of an interactive software system, with each constraining the design of the other:

1. **there is a need for the SE and UE life cycles to be *connected* to support communication among roles between the two life cycles,**

and with respect to the pedagogy,

2. **there is a need for appropriate computer science curricula to train the SE and UE roles about the connections between the two processes.**

In particular:

1. There is a lack of project-development environments for interactive software that connect the SE and UE life cycles, thereby missing out on critical communication about the various relationships between the two processes from occurring during development within a project environment. This lack of communication in turn results in the deficiencies of:

a. **coordination** of independent development activities where usability and software engineers could benefit by working together on role-specific activities;

b. mechanisms to identify, understand, and enforce **constraints and dependencies** within and between the two life cycles;

c. **synchronization** of dependent development work activities and resulting work products to maintain consistency; and

d. mechanisms to **anticipate and react to change and insights** within and between the two life cycles.

2. There is a lack of appropriate computer science (CS) curricula to train the SE and UE roles about the dynamics of interactive-software development. This in turn results in the lack of:

   e. understanding of **expertise and role distinction** required for interactive-software development;

   f. **negotiation and feasibility analysis** skills required in the context of interactive-software development; and

   g. an awareness of **differing timelines and work products necessary** in the overall development process.

1.3 Background

In the following sections we provide a discussion on similarities, differences, current practices in interactive-software development, and perspectives of quality to provide context for this dissertation work.

1.3.1 Similarities between life cycles

At a high level, UE and SE share the same objectives:

- describing a life cycle of development processes, activities, and work products;
- seeking to understand the customer's and users' wants and needs;
- translating these needs into system requirements;
- designing a system to satisfy these requirements; and

- testing to help ensure their realization in the final product.

At the process level, both life cycles have similar stages such as requirements analysis, design, and specifications even though these stages entail different philosophies and practices as discussed in the next section.

1.3.2 Differences between life cycles

The objectives of the SE and UE are achieved by the two developer roles using different development processes and techniques. At a high level, the two life cycles differ in the requirements and design phases but typically converge into one at the implementation stage (Figure 3).

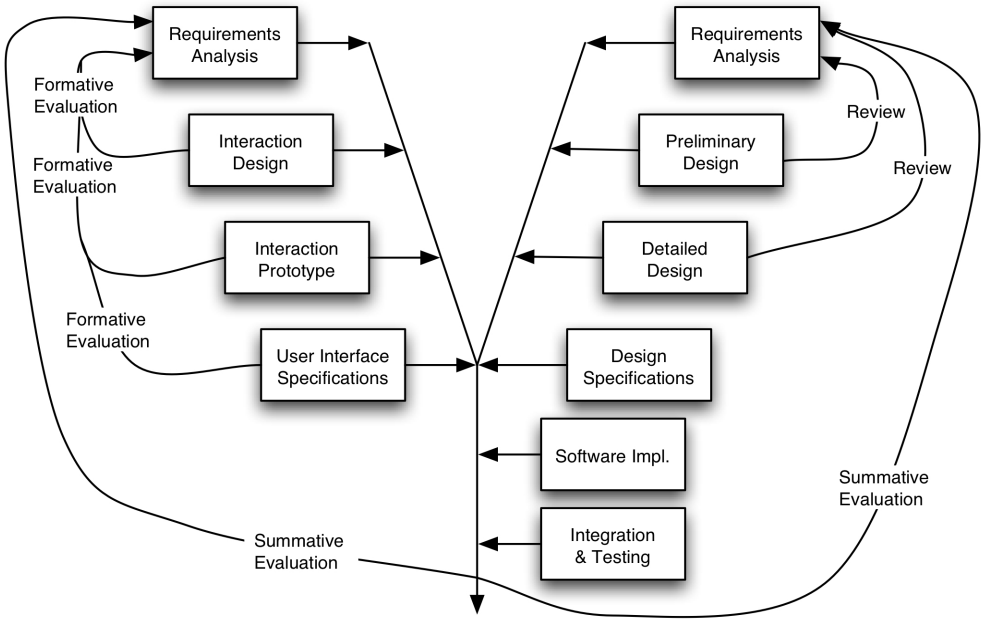


Figure 3: Current practices – SE and UE processes without connections

This convergence represents a point where these two processes are no longer completely parallel. There is a certain asymmetry between the two life cycles at this point because the SE role is ultimately responsible for the implementation of the UE design (Figure 4).

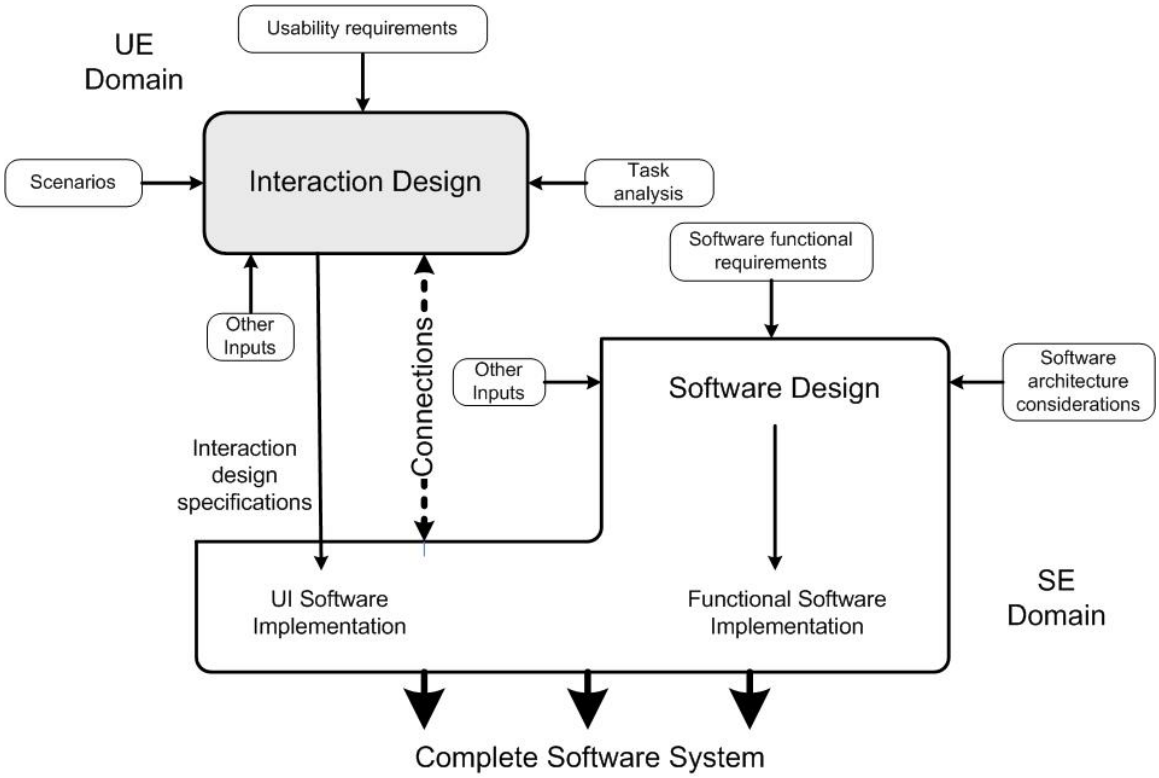


Figure 4: Asymmetry between the SE and UE life cycles

At each stage, the two life cycles have many differences in their activities, techniques, timelines, iterativeness, scope, roles, procedures, and focus. Several of the salient differences are identified next.

1.3.3 Different levels of iteration and evaluation

Developers of interaction designs often iterate early and frequently with design scenarios, screen sketches, paper prototypes, and low-fidelity, roughly-coded software prototypes before much, if any, software is committed to the user interface. Often this frequent and early iteration is done on a small scale and scope, and primarily as a means to evaluate a part of an interaction design in the context of a small number of user tasks. Usability engineers evaluate interaction designs in a number of ways, including early design walkthroughs, focus groups, usability inspections, and lab-based usability testing. The primary goal is to find usability problems or flaws in the interaction design, so the design can be iteratively improved.

Software engineers identify the problem, decompose and represent the problem in the form of requirements (requirements analysis block in Figure 2), transform the requirements into design specifications (preliminary and detailed design blocks in Figure 2), and then implement those design specifications. In the early days of software engineering, these activities were often performed using the sequential waterfall model (Royce, 1970). Later, these basic activities were incorporated into more iterative processes such as the spiral model (Boehm, 1988) (which has a risk analysis and an evaluation activity at the end of each stage). Even though the more recent SE development life cycles are evolving towards the UE style by anticipating and accommodating changes at each iteration, they still stress iteration on a larger scale (courser granularity) and scope. Moreover, testing and validation, which ensures integration accuracy and conformance to system specifications, are performed more towards the end of the development process and can include software for both the functional core and the user interface.

1.3.4 Differences in terminology

Even though certain terms in both life cycles sound similar they often mean different things. For example:

- In SE, the output of the design phase is a detailed specification document, usually in English, describing the structure and elements of the software system being built. These design specifications are then translated into code in the implementation stage. In UE the design phase often results in a design "specification" in the form of a prototype (not usually called "specification" in the UE domain) with the appropriate look-and-feel and a simulation of the behavior. This prototype is then handed over to the software developers for implementation in accordance with a "style guide" (also developed by the UE role) that describes the standards of the user interface elements for that organization. The term "specification," when used in UE domain, is generally used in the context of usability specifications. These specifications describe the target usability goals and the benchmarks for the system and have little to do with the implementation of the code.

- In UE, "testing" is a part of design, and is diagnostic in nature, being used to find and fix problems in the interaction design (identified as formative evaluation in Figure 1). Summative testing in UE is usually confined to research, but can be used to establish claims for product marketing. In SE "testing" is an independent stage where the objective is to check the implementation of the system and to validate its conformance to specifications. Analysis and verification of the design specifications performed in SE is often called "review" (identified in Figure 2) or verification and validation (V & V) (Boehm, 1984; Wallace and Fujii, 1989; Krauskopf and Rash, 1990; Lewis, 1992; Rakitin, 2001). When the specifications pass the review or V & V, they become a binding document between the client and the development team.
- Scenarios in SE (called "use-cases" in the object oriented design paradigm) are used to "identify a thread of usage for the system to be constructed (and) provide a description of how the system will be used" (Pressman, 2001). Whereas in UE, a design usage scenario is "a narrative or story that describes the activities of one or more persons, including information about goals, expectations, actions, and reactions (of persons)" (Rosson and Carroll, 2002).
- The SE group usually refers to the term "develop" to mean creating software code, whereas the usability engineers use "develop" to mean iterate, refine, and improve usability to create an interaction design.

Overall, the software engineers concentrate on the system whereas the usability engineers concentrate on the users. This fundamental difference in focus is one of the reasons why it is difficult to connect these two life cycles.

1.3.5 Differences in requirements representation

Most requirement specifications documented by software engineers use plain English language and are generally very detailed. These specifications are written specifically to drive the SE development process. On the other hand, usability engineers specify interactive component issues such as feedback, screen layout, colors, etc. using artifacts like prototypes, design scenarios, and screen sketches. These artifacts are not detailed enough to derive software design, instead they require additional refinement and

reformulation before implementation. Therefore, they cannot be used to directly drive the software development process. Later on, these UE artifacts, after rigorous formative evaluation, become requirements for the UI software component.

1.3.6 Change management

On one end of the spectrum of software life cycles in practice today are the documentation intensive, static, and "ironbound" contractual efforts. The ponderous weight of voluminous static documentation inherent in these types of projects do not allow effective mechanisms to predict or accommodate the effects of change, especially changes that occur very rapidly in early stages of a life cycle. It can be argued that configuration management processes (Joeris, 1997) and traceability techniques that exist in SE are an exception to this. Configuration management tools provide mechanisms and procedures to track changes in the work artifacts generated in a software development life cycle. Traceability mechanisms, often applied to requirements engineering stages in software engineering, allow developer roles to analyze and follow change propagation in a requirement specification document of a software system. However, these tools and techniques were mostly developed for SE life cycles; and do not address the affects of change between the two life cycles. In addition, these tools and techniques do not tackle change prediction and early warnings.

At the other end of the spectrum, many project managers use intensively hands-on project management principles wherein a project leader walks around managing and communicating with the various developers in a direct "hands-on" manner taking individual responsibility to make sure all the details and changes are addressed. This approach is based on the potential effectiveness of an informal and low-documentation approach to software development and the fact that a skilled human manager can keep track of changes and what needs to be done better than an automated system. However, this approach does not scale-up well as projects get more complex and if the manager has to keep track of all details of changes in both life cycles of a very large project as it progresses.

1.3.7 Quality in the development of interactive systems

Interactive-software development is a complex undertaking requiring a variety of analysis, design, development, and evaluation techniques and involving a diverse group of stakeholders. Most of these techniques are, in turn, practiced as part of smaller sub-processes; for example, interactive-software development requires two main life cycle processes of SE and UE, each of which in turn encompasses sub-processes such as requirements engineering life cycles and verification and validation in SE, and systems analysis and usability evaluation in UE. Each of these component sub-processes requires different developer roles with different skill-sets to build a single interactive software system within the constraints of a project. Apart from the different developer roles, there are other important stakeholders in the project such as end users, clients, and support and maintenance personnel.

With so many entities and stakeholders interacting within the overall project space to ultimately create and use a single system, each with its own perspectives on quality, it is difficult to define quality as pertaining to the total interactive software system without running the risk of being vague or incomplete. The obvious reason for this difficulty is due to the fact that quality manifests in different ways to different people and therefore the key to analyzing quality of a software system is usually dependent on the perspective of the party involved. For the end users, a system of high quality is most probably one with high measures of usability. For a software developer, a system of high quality could potentially be one which conforms to all stated requirements while at the same time has low complexity. For users, it might be about getting those requirements right in the first place. Similarly, for a maintenance role quality could manifest in terms of reusability.

1.3.7.1 Perspectives on software quality

A survey of the literature finds many definitions of software quality that cater to the perspectives of select stakeholders. For example, according to Pressman, quality is "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software" (2001). He acknowledges that this definition can be modified, extended, and debated "endlessly" and proceeds to emphasize three points

pertaining to quality: conformance to explicit software requirements as the foundation of quality, adherence to standards that define the development criteria for engineering software, and conformance to implicit requirements such as "the desire for ease of use and good maintainability". It is not surprising that this definition emphasizes software requirements and treats usability (if we consider the reference to "ease of use" as a reference to the much broader concept of usability) as an implicit requirement, given the fact that Pressman is a software engineer and therefore has a software engineer's perspective.

1.3.7.2 Quality in the context of interactive-systems development

Providing a definition of software quality that encompasses all aspects of a software project space is beyond the scope of this work. In this work we take an approach to defining software quality that is more appropriate and encompassing of the various entities involved in an interactive-systems development effort. At a general level, we believe, the issue of quality in a given development effort can be described using two high-level perspectives: quality as pertaining to the development process and quality as pertaining to the developed products resulting with the application of the process. Applying this definition to the development space of interactive systems, with component SE and UE life cycles, theoretically speaking we have four major perspectives for quality as shown in Figure 5: SE process, SE product, UE process, and UE product quality. Using this approach to quality, the definition provided by Pressman, as described in the previous paragraph, falls mostly on the SE product part on the top right of Figure 5 ("conformance to explicitly stated functional and performance requirements [and having] implicit characteristics that are expected of all professionally developed software") with the reference to "explicitly documented development standards" falling on the SE process part on the top left. In practical terms, given that there is only one product, in the form of an interactive software system that is finally deployed to the customer, the SE and UE products are one and the same. Also, the quality of SE and UE products are tightly coupled, and impact one another to a large extent (Bass and John, 2003).

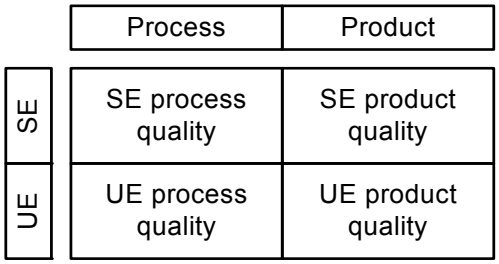


Figure 5: Quality types in the context of interactive-software development

1.3.7.3 The process dimension of quality

On the process dimension of software quality (left column in Figure 5), there exist significant amount of literature on the SE part (top left in Figure 5). For example, at an organizational level the quality of a process can be measured indirectly using the maturity level achieved by that process. One such SE process maturity model is the Capability Maturity Model (CMM) (Paulk, et al., 1993) which specifies levels such as "initial" where the process is loosely (if at all) defined and success is dependent on individual knowledge, skill, experience, and initiative, "repeatable" where the basic scheduling and cost analysis practices are in place and these practices are repeatable, "defined" where the practices are standardized, documented and integrated into an organization-wide practice, "managed" where process quality metrics (as defined by CMM) are collected and used quantitatively to control the process and products of a development effort, and "optimizing" where a development effort is constantly calibrated and improved by experimentation and introduction of new technologies (Pressman, 2001). The UE life cycle, being much younger, does not have as well defined process-based quality attributes as its SE counterpart. It is only recently that some work is being done in adopting the CMM-style process maturity models for UE life cycles (Allgood, 2004).

In this work, instead of taking the CMM-style standards approach, we define SE and UE process quality, in utility terms, as a measure of how well the various activities, phases, and timelines described in a life cycle process work for a particular development effort. We opine that a good process provides the right amount of flexibility and guidance to developer roles in accomplishing the goals and objectives of the project within a given set of constraints by expending the lowest possible effort, accommodating changes, and

affording mechanisms to contain and minimize surprises that often have costly side-effects.

Extending this utility-based definition of software process quality to interactive-software development, we characterize the quality of the overall software process as the measure of how well it supports the joint execution of SE and UE life cycles by providing mechanisms for communication (in the forms of coordination, constraint and dependency checking, synchronization, and change management). In other words, we take the perspective that the more a particular process affords these factors, the better the quality of that process.

1.3.7.4 The product dimension of quality

The other important aspect of a development effort is the product that is created as a result of the application of a process. The term product does not necessarily mean the finished software system alone but also refers to work products that are created *during* the different phases of the process (e.g. Software Requirements Specifications, user interface prototypes, benchmark task descriptions). The ultimate goal of a development process is to create a quality product, and the quality of a product is often dependent on the quality of the process used to create that product.

Some of the most commonly discussed software quality attributes such as conformity (to requirements), reliability, maintainability, complexity, reusability, and usability are product-oriented. Out of these, except for usability, all attributes fall in the SE product category on the top right portion of Figure 5. Similarly, even though not widely discussed, the broad concept of usability has a variety of product-based attributes (bottom right part of Figure 5) such as ease-of-use, user satisfaction, learnability, retention over time, and efficiency of task performance.

1.3.7.5 Measuring quality

It is difficult, if not impossible, to directly measure the quality of a process or a product. Because of this, quality is usually measured using indicators pertaining to the attributes of a process or a product. For example, the quality of a product can be estimated using measures such as usability and complexity. Similarly, the quality of a process can be

estimated by measuring the cost per project module completed or the level of maturity on a CMM scale. Details of the evaluation setup, metrics, and procedure for the study in this research are provided in Chapter 6.

1.4 Current Practices and Motivation

In spite of the extensive research and maturity levels achieved individually in areas of UE and SE life cycles, there has been a marked deficiency of understanding about connections between the two. Underlying this lack of understanding is fact that the corresponding developer roles, in general, do not identify with the other's goals and needs and do not have an appreciation for the other's area of expertise. One apparent reason for this deficient understanding and appreciation is the way computer science courses are typically offered in colleges and universities: SE courses often omit any references to usability and user interaction development techniques, and UE courses often do not discuss the SE implications of usability patterns (Douglas, et al., 2002), (Pyla, et al., 2004b).

Compared to software engineering life cycle concepts, which have been in existence for much longer periods of time, usability engineering life cycle concepts are significantly younger and less extensively adopted in most software development companies. A result of less maturity of usability engineering combined with the lack of understanding and appreciation of its importance often results in the usability engineering life cycle not being practiced in its entirety or, worse, ignored completely. Usability engineers are often brought in *after* the software components are completed and asked to *fix* the usability of the system. At this stage in the development life cycle any usability fix or recommendation that requires architectural or major backend modification is too expensive to make and is often left out. Only minor or cosmetic changes are addressed, with the potential of having serious usability problems in the developed system.

Given that both SE and UE development life cycles are now reasonably mature and well established, both have the same high-level goal of producing software that the user wants and needs, and both recognize that the two must function together to create a single system, one might expect well-defined connections for collaboration and communication

between the two development processes and for the resulting overall software system, backend plus the UI, to have high quality. However, for the most part, the two disciplines are still considered as separate and are applied independently with little coordination during product development. The result is a system falling short in both quality of the product (functionality and usability) and in some cases the process (completely failed project) itself (The Standish Group, 1994; The Standish Group, 1995; The Standish Group, 1999; The Standish Group, 2001).

With the ever-increasing use of software systems today and the growing amount of resources being expended in software development, there is a need to investigate why the overall quality of systems (product) with interactive components is not as good as it could be, identify potential factors that affect the overall quality of software development (process), and investigate if and how these factors affect the overall quality.

Based on our review of the literature, informal interviews with software developers in the real-world, and brainstorming with researchers and experts, it appears that in order to collaborate successfully on a single cohesive final product, the SE and UE life cycles should be *connected* and that each of the UE and SE roles must work with the other role. More specifically, we have identified the following factors that appear to be prime candidates for impacting quality in an interactive-software development effort. In the next few sections we provide rationale as to why we selected these factors and how they may impact quality.

1.4.1 Communication among different life cycle roles

Although the two life cycle roles can successfully do much of their development independently and in parallel, because of the tight coupling between the backend and the UI, a successful project requires that the two roles communicate so that each knows generally what the other is doing and how that might affect its own activities and work products. The two roles cannot collaborate without communication and the longer they work without knowing about the other's progress and insights, the greater their work diverges, and the harder it becomes to bring the two life cycle products together at the end.

Communication is important between the SE and UE roles to have activity awareness about how the other group's design is progressing, what development activity they are currently performing, what features are being focused on, what insights and concerns they have for the project, what directions they are taking, and so on. Especially during the early requirements and design activities, each group needs to be "light on its feet" and able to inform and respond to events and activities occurring in the counterpart life cycle. However, current practices (Figure 3) do not permit that necessary communication to take place because the two life cycles operate independently; that is, there is no structured development framework to facilitate communication between these two life cycles, leaving cross-domain (especially) communication dependent on individual whim or chance. Based on our real-world experience, day-to-day communication processes have proven to be inadequate and often result in nasty surprises that are revealed only at the end when serious communication finally does occur. This is known to be often too late in the overall process.

One might argue that the communication process need not be more formal than it is right now and that the usability and software engineering practitioners should be on the same analysis team. Indeed, in their day-to-day life, the two developers are often technically on the same analysis team. But our real-world experience has shown that this is not enough to foster the necessary communication (especially about features and changes) because each role still focuses almost completely on their own problems and their own designs. For example, the SE role in general is not concerned about UE role's interaction design and vice versa. So our communication focus is not on formality, but on completeness.

In this work we have identified the following forms of communication to be candidates that have the potential to affect the quality of an interactive-software development endeavor.

1.4.1.1 Coordination of development activities

When the two life-cycle concepts are applied in isolation, the resulting lack of understanding between the two developer roles, combined with an urgency to get their

own work done, often leads to working without collaboration (Figure 3), when they could be more efficient and effective coordinating with one another.

The lack of coordination between the usability and software engineers often results in not getting the usability needs of the system represented in the software design. This usually leads to conflicts, gaps, design and requirements mismatches, miscommunication, "spaghetti" code, and brittle software due to unanticipated changes and other serious problems during development.

Without coordination, the two roles duplicate their efforts in usability and software engineering activities when they could be working together. For example, both the SE and UE roles conduct field visits and client interviews for systems analysis and requirements gathering during the early stages of system development. Software engineers elicit functional requirements (Pressman, 2001), and determine the physical properties and operational environments of the system (Lewis, 1992), etc. Usability engineers visit clients and users to determine, often through "ethnographic studies" (Blomberg, 1995), how users work and what computer-based support they need for that work. They seek task information, inputs for usage scenarios, and user class definitions. Often, the activities of the two roles are performed independently, missing opportunities for team building, communication, presenting a unified developer "face" to clients, and early agreement on goals and requirements. Why not coordinate this early systems analysis effort? There is potential for much value to be derived from cooperative system analysis and requirements gathering. Such joint activities help in team building, communication, and in each life cycle role recognizing the value, and problems, of the other, in addition to early agreement on goals and requirements. In addition, working together on early life cycle activities is a chance for each role to learn about the value, objectives, and problems of the other. Instead, each development group reports its results in documentation not usually seen by people in the other life cycle. Each uses those results to drive only their part of the system design and finally merge at the implementation stage (Figure 3), where it is much too late to discover the differences, inconsistencies, and incompatibilities between the two parts of the overall design. Moreover, this lack of coordinated activities presents a disjointed appearance of the

development team to the client. It is likely to cause confusion in the clients: “why are we being asked similar questions by two different groups from the same development team?”

Another significant shortcoming of the practice shown in Figure 3 is the fact that the independently generated user interface specifications on the UE side and functional design specifications on the SE side are submitted to the development team at implementation stage. However, because these specifications were developed without coordination and communication, when they are now considered together in detail, developers typically discover that the two design parts do not fit with one another because of large differences and incompatibilities.

1.4.1.2 Constraint mapping and dependency checks

Because each part of an interactive system must operate with the other, many system requirements have both SE and UE components. If SE component or feature is first to be considered, it should trigger (or be mapped to) a reminder that a UE counterpart is needed, and vice versa. When the two roles gather requirements separately and without communication, it is easy to capture requirements that are conflicting, incompatible or one-sided. Even if there is some ad-hoc form of communication between the two groups, it is inevitable that some parts of the requirements or design will be forgotten or will “fall through the cracks”.

The lack of understanding of the constraints and dependencies between the two life cycles’ timelines and work products often create serious problems such as inconsistencies in the documentation (work products) of the SE and UE design. As an example, software engineers perform a detailed functional analysis from the requirements of the system to be built. Usability engineers perform a hierarchical task analysis, with usage scenarios to guide design for each task, based on their requirements. Documentation of these requirements and designs is maintained separately and not necessarily shared. However, each view of the requirements and design has elements that reflect counterpart elements in the other view. For example, each task in the task analysis on the UE side implies the need for corresponding functions in the SE specifications. Similarly, each function in the software design can reflect the need for access to this functionality through one or more

user tasks in the user interface. Without the knowledge of such dependencies, when tasks are missing in the user interface or functions are missing in the software because of changes on either life cycle, the respective sets of documentation have a high probability of becoming inconsistent.

Constraints, dependencies, and relationships exist not only among activities and work products that cross over between the two life cycles but also within each of the life cycles. For example, on the UE side, a key task identified in task analysis should be considered and matched later for a design scenario and a benchmark task. To our knowledge, there are no existing life cycle frameworks that help in addressing such internal and external constraints, dependencies, and relationships among life cycle activities.

In general, design choices made in one life cycle constrain the design options in the other. In our consulting experience we often encountered situations where the user interfaces to software systems were designed from a functional point of view and the code was factored to minimize duplication on the backend core. The resulting systems had user interfaces that did not have proper interaction cues to help the user in a smooth task transition. Instead, a task oriented approach would have supported users with screen transitions specific to each task; even though this would have resulted in a possibly "less efficient" composition for the backend. Another case in our consulting experience was about integrating a group of individually designed web-based systems through a single portal. Each of these systems was designed for separate tasks and functionalities. These systems were integrated on the basis of functionality and not on the way the tasks would flow in the new system. The users of this new system had to go through awkward screen transitions when their tasks referenced functions from the different existing systems. Because of the constraints on one another, independent application of the two life cycles (Figure 3) is likely to fail.

1.4.1.3 Synchronization of development schedules

In current practices, the life cycle roles must synchronize the work products eventually for the implementation and testing phases. However, waiting until one absolutely must

synchronize obviously creates problems. Therefore, it is better to have many synchronization points, earlier and throughout the development life cycle. These timely synchronization points would allow earlier, more frequent, and less costly "calibration" to keep both design parts on track for a more harmonious final synchronization with fewer harmful surprises.

The lack of synchronization of the design work products of the two life cycle roles, which is usually put off until the implementation and testing phases, near the end of the development effort creates big surprises that are often too costly to address. For example, as mentioned in the current practices section, it is not uncommon to find usability engineers being brought into the project late in the development process, even after the SE implementation stage. They are asked to test and/or "fix" the usability of an already-implemented system, and then, of course, many changes proposed by the usability engineers that require significant modifications must be ignored due to budget and time constraints. Those few changes that actually do get included require a significant investment in terms of time and effort because they must be retrofitted (Boehm, 1981). Also, often during the development life cycle, there is a need for synchronizing the timing of activities in such a way that there is timely readiness of work products when the other development role needs them. This prevents situations where one development role must wait for the other one to complete a particular work product. However, as shown in Figure 3, the more each team works without communication and collaboration, the less likely they will be able to schedule their development activities to arrive simultaneously at common checkpoints.

1.4.1.4 Provision for anticipation and reaction to changes and insights

In the development of interactive systems, each phase and each iteration has a potential for change. In fact, at least the early part of the UE process is intended to change the design iteratively. This change can manifest itself during the requirements phase (growing and evolving understanding of the emerging system by developers and users), design stage (evaluation identifies that the interaction metaphor was not easily understood by users), etc. Such changes often affect both life cycles because of the

various dependencies that exist between and within the two processes. Therefore, change can conceptually be visualized as a design perturbation that has a *ripple* effect on all stages in which previous work has been done. For example, during the usability evaluation, the usability engineer may recognize the need for a new task to be supported by the system. This new task requires updating the previously generated hierarchical task analysis document and generation of new usage scenarios to reflect the new addition (along with the rationale). This change to the HTA generates the need to change, on the SE side, the functional decomposition (by adding new functions to the functional core to support this task on the user interface). These new functions, in turn, mandate a change to the design, schedules, and in some cases even the architecture of the entire system. Thus, one of the most important requirements for system development is to identify the possible implications and effects of each kind of change and to account for them in the design accordingly. Another important requirement is to try to mitigate the impact of change by communicating about changes as early as possible, and by directing that communication directly to the development activities most affected. The more the two developer roles work without a common structure (Figure 3) the greater the possibility that inevitable changes in each part will introduce incompatibilities, revealed as "surprises" when they finally do communicate.

One particular kind of dependency between life cycle parts represents a kind of "feed-forward", giving insight to future life cycle activities. For example, during the early design stages in the UE life cycle, the usage scenarios provide insights as to how the layout and design of the user interface might look like. In other words, for development phases that are connected to one another (in this case, the initial screen design is dependent on or connected to the usage scenarios), there is a possibility that the designers can forecast or derive insights from a particular design activity. Sometimes the feed-forward is in the form of a note: "when you get to screen design, don't forget to consider such-and-such." Therefore, as and when the developer encounters such premonitions or ideas about potential effects on later stages (on the screen design in this example), there is a need to document them when the process is still in the initial stages (usage scenario phase). When the developer reaches the initial screen design stage, the previously documented insights are then readily available to aid the screen design activity. To our

knowledge, none of the current approaches to the development of systems with interactive components provide this capability.

1.4.2 Training SE and UE roles within computer science curricula

One of the most important reasons why SE as a field has failed to connect to the UE discipline is that the processes and activities in SE have their roots in non-interactive, batch processing based, systems-side applications development. On the other hand, one plausible explanation for why the UE discipline failed to connect to the SE domain is a lack of knowledge on the part of the usability engineers toward SE processes, constraints and schedules. Overall, the developers from these two disciplines have very little understanding and appreciation for each other's skills, techniques and backgrounds. One fundamental reason for this state-of-the-art is the lack of education curriculum in CS that address the various issues involved in the development of interactive software.

Traditionally, the concepts, theory and techniques associated with the user interface (UI) design domain are taught as a part of Human Computer Interaction (HCI) courses in most computer science departments. These courses use text books such as (Newman and Lamming, 1995; Shneiderman, 1998; Rosson and Carroll, 2002) and cover topics that are completely focused on the user interface side of system development. Little or no mention is made of the required SE role to implement the UI specifications. The SE courses as they are offered in most computer science departments are no better than their HCI counterparts when it comes to addressing the connections between the two domains. SE courses often omit any references to UE role required to design the UI.

In this work we have identified the following pedagogical topics to be candidates for training SE and UE roles in the context of interactive-software development.

1.4.2.1 Expertise and role distinction

Most existing CS curricula do not make clear the distinct roles that are involved in the development of interactive software. For example, on the UE side, students who complete HCI courses that make no reference to the counterpart SE role gain little, if any, appreciation for the development process associated with the functional part of the

system. They may wrongfully believe that any usability or user interface specification derived by them will "somehow" get automatically implemented by "someone", which in reality is rarely the case. To the contrary, simple usability features, such as undo, that the students might take for granted, can have serious architectural implications in the functional part of the system. Similar problems exist on the SE side as well. We surveyed a majority of the top selling books in software engineering/software design and engineering (according to facultyonline.com website) (2007b). Out of these, only a small fraction of textbooks identify and dedicate sections to existence of a parallel life cycle for UE.

1.4.2.2 Negotiation and feasibility analysis

Because of the tight coupling between UI features and the functional core (Bass and John, 2003), it is essential that the two roles frequently communicate to negotiate and ascertain the compatibility of their individual designs. Most HCI or SE curricula fail to address this important aspect of interactive-software development. For example, Rosson and Carroll (2002) talk about scenarios and focus on the UE life cycle and process. They do not provide a clear description of the two development efforts and the implications for the functional core. The incremental case study provided in this book proceeds through different interface design stages (as the chapters progress), and ends up with a system entirely developed and including user documentation. The readers are not informed how the user interface specifications are communicated to the software developers, nor how the design constraints are negotiated between the two development bodies. Same problems exist on the SE side. The few SE text books that identify and dedicate sections to UI concerns still do not address the dependencies and implications of the two processes on one another. Students completing software engineering courses that follow the syllabus in textbooks like these obviously will gain little or no appreciation for the user interface issues and the constraints on the functional core that may arise due to them.

1.4.2.3 Timelines and work products

Most HCI courses stress the importance of iterativeness in the UE life cycle and advocate a light-weight development process. SE courses also promote iterativeness albeit to a lesser degree. However, what is not discussed in these classes is how these differences in

iterativeness are impacted when the two roles are working together. For example it does not help the overall process if the UE role is in the middle of an iteration when the SE role enters their design phase. Another aspect that is rarely covered in traditional SE and UE courses is the need for each role to share and discuss one another's work products to ensure their designs are consistent with one another.

1.4.2.4 Strategies for change management

One of the key constants in interactive-software development is change. Because of the complexity involved in designing systems that fit the needs of different stakeholders it is almost impossible to get the design right the first time. Therefore the two roles often employ prototyping and other techniques to manage change inexpensively. Implementation of the "real" product (a high-fidelity prototype in the case of the UE life cycle) is only undertaken after the design has stabilized enough for each role; even then the respective products are often changed as each life cycle progresses. This problem of change management is further aggravated in when two life cycles have to work together. Not only should each role manage change in their life cycle but should also design and react to change in the counterpart life cycle. Current HCI and SE curricula do not address these issues.

1.5 Research Goals

At a high-level we had two research goals to address the two parts of the problem statement (Section 1.2) respectively:

1. Connect the SE and UE life cycles by creating a project development environment that fosters communication between the SE and UE life cycles, and explore the question of whether and in what way the following four communication factors affect the quality of development of interactive software systems:

   a. **coordination** of development activities that have commonalities between the two life cycles' processes;

   b. **constraint and dependency enforcement** between the two life cycles' phases and work products;

c. **synchronization** of dependent development activities; and

d. **anticipation and reaction to change** within the overall development effort.

2. Connect the SE and UE life cycles by creating a graduate-level cross-pollinated SE-UE joint course offering to explore whether and in what way the following factors influence the effectiveness in educating SE and UE roles about the intricacies of interactive-software development:

a. **expertise and role distinction** in the context of working with SE and UE roles;

b. **negotiation and feasibility analysis** among SE and UE roles about UI and functional features and design; and

c. **timelines and work products** pertaining to the SE and UE life cycles.

1.6 Research Approach

In this dissertation we approached our research goals in two ways, by:

1. Creating the Ripple project development environment to foster communication between the SE and UE roles. This was accomplished by:

a. creating an abstract formal description of the total "development space" in order to precisely and completely express the properties and attributes of various entities involved in an interactive-systems development effort and the constraints and dependencies within them. This description, called the Ripple Description Model (RDM), consists of an abstract formal notation to describe the structure, activities, work products, developer roles, timelines, connections, and interactions between the two life cycles using concepts from discrete mathematics and set theory,

b. rendering the RDM more concrete, in the form of the Ripple Implementation Framework, which describes specifically how the RDM could translate into a software development environment system, including tool support, entities, and various components used in the development of interactive systems. This framework is expressed at a level of detail of the major software components to support work product repository, the constraint module, and the life cycle

descriptions. This framework would provide the tools to specify and represent different types of life cycles, development approaches, development techniques, and project constraints, but does not stipulate what form each of these items must take. Therefore this framework can be *instantiated* to apply to a particular project; we call the result of such an effort a Ripple Implementation Instance. This implementation instance provides the developers the flexibility to adopt activities, techniques, and supporting systems to support the needs of a particular development effort with its own constraint set,

c. creating a Ripple Implementation Instance to represent the development of an interactive system in a classroom setting, and
d. evaluating how the above mentioned communication factors affect the quality of the SE and UE processes and products by using the classroom instantiation.

2. Creating a graduate-level cross-pollinated SE-UE joint course-offering to educate SE and UE roles about the intricacies of interactive-software development. This was accomplished by:

a. demonstrating a proof of concept of a cross-pollinated graduate-level joint SE-UE course in the Department of Computer Science at Virginia Tech through which we trained students from UE class to perform the UE role and students from SE class to perform the SE role in undertaking semester-long joint team projects,

b. simulating different conditions of interactive-software development (i.e. with different types of project constraints and role playing explained in Section 6.3.1) and assigning different teams to these conditions. These teams developed prototype systems for a real client using their assigned development condition, and

c. requiring the teams to record experiences and lessons learned as they use the assigned development condition. Each team shared the knowledge they gained and lessons learned to teams from other conditions via an end-of-semester research symposium. Different instruments were used to measure how the above

mentioned pedagogical factors affect the quality of the cross-pollinated SE-UE curricula.

In other words, the research approach adopted in this dissertation could be summarized as:

1. Investigate the effects of communication factors on quality by:
   a. Creating the Ripple Description Model to formally and abstractly chart the interactive-systems development space,
   b. Rendering the Ripple Description Model concrete by creating a Ripple Implementation Framework to describe how the Ripple Description Model would translate into a real software system supporting the development environment, roles, and entities present in an interactive-systems development effort,
   c. Generating a Ripple Implementation Instance of the Ripple Description Model to suit the requirements and constraints of a development effort in a classroom setting, and
   d. Investigating how communication factors affect the quality of the SE and UE processes and products by using this classroom instantiation.
2. Investigate the effectiveness of a cross-pollinated SE-UE curriculum by:
   A. Creating a cross-pollinated SE-UE course by offering graduate-level SE and UE classes together and assembling teams from across the two classes to play the UE and SE roles,
   B. Simulating different conditions of interactive-software development in a classroom setting and assigning different teams to each of these conditions, requiring them to conceive, design, and develop a software system, given the same project constraints,
   C. Requiring the teams to record experiences and lessons learned as they use the assigned development condition, and demonstrating these lessons learned to teams in other conditions via an end-of-semester research symposium.

1.7 Scope and Limitations

In this work, we explored the relationships between quality (of process and product) and the factors of communication such as coordination, constraint and dependency checking, synchronization, and anticipation and reaction to change. We conducted an evaluation of an instance of the Ripple Implementation Framework by creating teams comprising of usability engineers and software engineers from graduate level UE and SE classes respectively. These teams were asked to develop a functional prototype of an interactive software system; some of these teams used a Ripple Implementation Instance and the others used a development environment that represented real-world practices where the above factors were controlled.

However, we were not able to distinguish the individual contributions of each factor because of the classroom setting, the lack of control on each individual factor, and the fact that this was a one-time opportunity to run the study. Clear differentiation of the contributions of each individual factor on an individual quality attribute requires conducting numerous separate studies and varying the measure of each factor while keeping all other factors constant. Another approach to differentiation is by controlling a single factor per set of groups in a large study and measuring the effects of that factor on the quality of process and product. Because of resource limitations and classroom constraints these two approaches to differentiation could not be adopted, resulting in our investigating the effects of the set of factors, taken as whole, on the various quality aspects. However, this study provides a foundation for future studies beyond this dissertation, where contributions of each individual factor on quality attributes of process and product can be studied.

1.8 Operating Assumptions and Constraints

Our research objectives for creating the Ripple Description Model and the Ripple Implementation Framework are guided by the assumptions that they:

- are not designed for or biased toward any particular software engineering life cycle process such as the waterfall model (Royce, 1970) or spiral model (Boehm, 1988), but are intended to connect to any SE process;

- are not focused on particular usability engineering life cycle approaches such as the Star Life Cycle (Hix and Hartson, 1993) or theoretical approaches such as the scenario-based design (Rosson and Carroll, 2002) but are intended to connect to any UE process; and
- do not merge the SE and UE development processes into a single process but retain them as separately identifiable processes, each with their own life cycle structure, development activities, and techniques.

Our research objective of whether communication factors such as coordination, synchronization, constraint and dependency mapping, and provision for anticipating and reacting to change within an overall SE and UE development environment have an effect on the overall quality of the process and product was also constrained. Namely, the quality of processes and products manifests in wide range of attributes such as usability, maintainability, reliability, complexity, and efficiency and it was not possible to measure all such attributes directly in a classroom-style evaluation. In Chapter 6 we discuss the evaluation setup, the attributes we measured, and the metrics used.

1.9 Research Contributions

Based on our work in this area, the following is a summary of our research contributions:

1. **Demonstrated feasibility of connecting SE and UE processes** by demonstrating a project-development environment that fosters communication to facilitate coordination of independent development activities, enforcing constraints and dependencies within and between the two life cycles, synchronization of dependent development work activities and resulting work products to maintain consistency, and to anticipate and react to change and insights. Specifically, the following contributions were made in this regard:

   a. **Charted the interactive-systems development space**: Using the Ripple Description Model, we were able to abstractly express and describe the interactive-systems development space, which includes the two life cycles, their constituent activities, work products, and the connections and dependencies within these different entities. This abstract representation later drove the

development of a framework that concretely defined the development environment and tool support necessary for interactive-systems development.

b. **Created a framework that describes software development environment for interactive systems:** By identifying major components such as work product repositories and mechanisms such as constraint enforcement that are necessary to connect SE and UE life cycles, we created a framework that fosters communication and improves quality of resulting work products. This framework has the flexibility to be used for any SE and UE life cycles and within any given set of project constraints.

c. **Identified effects of key communication factors on process and product quality**: By instantiating the Ripple Implementation Framework in a classroom setting and evaluating it in an exploratory study, we were able to identify the effects of communication in the form of coordination, constraint and dependency checking, synchronization, and change management on the quality of the development process and resulting products.

d. **Laid foundation for a software environment to support SE and UE connections**: With the Ripple Description Model, the Ripple Implementation Framework, and the Ripple Implementation Instance, we were able to understand the development space of an interactive-software development effort. With the insights gained during our evaluation, we now have a reasonable understanding of the various components a project management environment should have for supporting the development of interactive software systems. With this understanding it is now possible to specify a software tool/environment that can support these entities.

e. **Performed detailed analysis of inter-role communication within an interactive-software development environment**: Using our exploratory study, we were able to conduct a detailed analysis of how different roles, personal and professional biases, group contexts, and project constraints impact the product and process in a software development environment. Based on these analyses we

derived implications for organizational and management level issues in the real-world.

2. **Demonstrated a proof of concept for cross-pollinated SE-UE curriculum** through a one-of-its-kind joint offering of SE and UE courses. Using this experimental course offering, we were able to provide a more effective learning environment for students enrolled in these classes. Specifically, the following contributions were made in this regard:

   a. **Introduced students to the different roles and expertise necessary for interactive-software development**: By training students in SE class for SE role and UE class for UE role and requiring students to work on semester-long team projects for a real client we facilitated learning of different roles by the students actually performing in those roles.

   b. **Demonstrated the dynamics and tradeoffs involved in different conditions of interaction between the SE and UE life cycles:** By simulating different interactive-software development conditions and requiring students to record and synthesize their experiences, and then by making them share these experiences, we were able to educate the students about the advantages and disadvantages in following different paradigms of interactive-software development.

   c. **Identified curriculum recommendations for teaching cross-pollinated SE-UE courses:** Based on our experience in offering graduate-level joint SE-UE classes we identified different pedagogical recommendations for CS educators who attempt to teach similar courses.

2 Chapter Two: History and Related Work

2.1 A True Story

It was an important day for one academic department on Virginia Tech campus because of the visit by a highly distinguished design scholar, the lead of one major academic and professional society, in the area of Human Factors and Ergonomics. The visitor was scheduled to give a talk at 2:00PM. The lecture hall was full with eager graduate students, researchers, and faculty members from HCI, human factors, and other design related areas waiting to hear the guest talk. One of the graduate students, probably the organizer of the event, confidently walks up to help the speaker "setup" her presentation on the computer. What happened in the next half hour was an interesting and insightful look into the current state-of-the-art of computing technology that surrounds us in everyday life.

Needless to say, the "setting up" of the presentation, which was stored on the speaker's USB flash drive turned out to be a much harder task than anticipated and very soon involved the expertise of about half a dozen others in the room. The presentation file could not be loaded from the USB drive after it was plugged into the computer. In the course of next several minutes, while trying to troubleshoot the problem, revelations were made: "Oh! This is Mac," frustrations were expressed: "Don't know what it is doing," aspersions were cast "I don't think Macs understand that," hypotheses were conjured: "I think you hit this button," and finally arbitrary solutions were proposed: "You should not try to load the file from the USB device. You need to copy the file to the local drive and then load it." Nobody knew why that is so, just as they did not know "how to get rid of that clock on the bottom of the screen" that was interfering with the presentation, but the idea worked. The clock was dragged as far into a corner as possible (a small part if it was still sticking out) and the slideshow was started.

The talk started 20 minutes late and the organizer apologized for the delay citing "technological problems." The speaker proceeded to give a highly informative and interesting presentation of her work on designing smart homes. The talk was periodically punctuated with annoying intrusions by the slideshow software whenever the speaker encountered high resolution images in her presentation.

Sitting in the audience we could not help but wonder at the fact that it took half a dozen bright design scholars, each an expert user of computers, half an hour to wrestle with technology to get a simple presentation running from a supposedly "universal" storage device. A deeper analysis of the occurrences in this story divides into two philosophical positions:

One, taking a systems perspective, today **software systems** make it possible (with effort) to combine a rich variety of information media into a presentation, use powerful visual effects to manipulate and disseminate that information in a presentation, carry that presentation around in a keychain sized device, "plug and play" with that device on almost any machine, and be able to use that information in a slideshow on even a different platform from the one it was prepared on.

Two, taking a users perspective, today **software users** are subjected to annoyances and frustrations on a daily basis when encountering computing technology. Instead of helping people *solve* problems it is becoming increasingly evident that computers *are* the problem (Landauer, 1995; Pew, 2003). If one were to reflect on the reasons for this state-of-the-art, one would realize that there are two aspects that seem to impact a user's encounters with technology: the capabilities of the system, and the mechanisms to access those capabilities.

We argue that, the quality of the functionality combined with the interaction experience provided by a computing system gives a measure of the philosophical distance between the two positions stated above. Today we have mature technologies and frameworks that independently afford greater functionality or better interaction experience but usually fail to provide both. This situation is a result of three main problems that often arise due to:

- Software engineering concerns: Software functionality not being quite right beyond the small scope it was developed for (e.g. the USB device drivers not working, as required, on the Macintosh platform);
- Usability engineering concerns: User interface controls not being intuitive and efficient (e.g. the inability to get rid of the clock from the screen); and most importantly
- SE and UE dependency concerns: Software functionality and interaction experience not developed in relation with one another (e.g. the system supports USB functionality and with the right affordances on the UI, lets the users to believe they can interact with the external drive like any other drive. The users can store presentation files and copy them to

the local disk, but cannot directly load the file from the external drive even though that matches the user's mental model better).

Why is the state-of-the-art like this? How did we arrive at this situation? What led to the two factors of functionality and interaction experience (and the principles to design for them) to mature independently? Were there any attempts to bridge this disconnect? If so, how and by who?

In this chapter we provide a historical background on computing with an emphasis on the two disciplines responsible for guaranteeing functionality and interaction experience, along with an interpretation in the context of the current state-of-the-art. We then provide a survey of related work in the literature for bridging the gaps between the user interface methodologies and their functional counterparts.

2.2 Historical Background

In order to understand and appreciate the progress (or reflect on the lack thereof) of computing as a means to *empower* humans, we argue that we need to know the roots, history, and factors that shaped the endeavors of computer scientists ultimately leading to the state-of-the-art. In this section, we use "computing" to mean an abstract discipline which includes hardware, software, personware, and the science and engineering principles related to them.

We have organized this history as a chronological sequence of eras. We acknowledge that history lends itself to be divided according to many criteria and that our work here is by no means exhaustive and complete enough to include all such criteria. We focus on those aspects that help put the gaps between SE and HCI in perspective. This organization is based on and adapted from the one provided by Pew (2003).

2.2.1 The pioneering years – 1966 and before

Target users of the era: The predominant users of this age were programmers. These were individuals who had to deal with punched cards on special machines to convert their logic into machine input.

Players and visionaries: Vannevar Bush ("As we may think" article (Bush, 1945)); J. C. R. Licklider ("Man-Computer Symbiosis" article (Licklider, 1965)); Douglas Englebart (invented

the mouse input device); Ted Nelson (invented "hypertext"); and Ivan Sutherland (pioneer of computer-aided design and graphic editing systems).

Technology: This age saw the beginnings of many hardware revolutions such as the cathode ray tube displays, memory drums, mouse, etc. Most of these technologies were in experimental stages at this time and programs were predominantly written in machine level assembly languages.

Context of computing: The main concerns of this age were issues such as how to format the outputs of the assembly level programs on punch cards within the constraints of a line printer with fixed column width and limited formatting capabilities. These efforts were predominantly towards making the debugging of programs easy and productive. The programmers most often had to work with "abstruse error messages" from unforgiving platforms that had zero margins for error. Those were the days when one extra space in a program would require the entire batch of punch cards to be resubmitted (Pew, 2003).

Context of SE and HCI: Even though they were not referred to as HCI and SE at that time, the major focus of this age was the creation of higher-level programming languages to help the dominant users of this time (programmers) become more productive. This age saw the birth of two such languages: COBOL (Common Business Oriented Language) and FORTRAN. COBOL used the English language and had documentation capabilities. FORTRAN was geared towards scientists and mathematicians and made possible to code complex mathematical constructs at a higher level. We argue that the "usability" of a programming task and the functional capabilities were the major concerns of this age. Philosophically we believe that this age probably saw the best connections between the high-level languages' user (programmer) interface and backend (programming constructs) until today. Using the terminology of today, we can say that the functional and interaction concerns were probably best connected during that time. One reason for these close connections between the UI and backend concerns is probably because they were not identified as separate domains during this time and that software developers were not yet aware of the need to address end-user concerns.

2.2.2 The time-sharing/structured programming years – 1967 to 1977

Target users of the era: The main users of this age were trained personnel working for huge organizations from domains such as insurance, defense, and the Social Security Administration.

Players and visionaries: Alan Kay (introduced parallel programming, windows, and message passing that would later become foundation to the OO paradigm); Allan Newell and Herbert Simon (psychology of HCI, human information processing ideas in HCI)

Technology: This age saw two important advances: one, the disappearance of small (relative to mainframe computers) independent machines and the start of the age of time-shared computers, and two, the emergence of interactive programming environments. It became possible to switch several individual users' programs and memory resources between fast magnetic core memory and comparatively slower but much larger memory on magnetic drums. The concept of remotely accessing a time-shared computing resource via a terminal device to perform online transactions became a reality. Advances in incremental compilers and other programming infrastructure including the ability to store computer programs in computer files made punch cards obsolete (Pew, 2003).

Context of computing: There were two main thrusts in computing in this era. One, huge organizations from domains such as insurance, defense, and the Social Security Administration started using online systems which contained all the information at a central location to perform "interactive" transactions from user terminals over telecommunication lines to various geographic locations. The delay between command and response was significant because of limitations on the speed of the hardware to access magnetic memories and the network latencies for remote transactions. This became an important concern and a prominent area of research on how such delays influence human performance was born. The "user interfaces" themselves were very limited and consisted of a few "screens" and usually had a "well-defined sequentially-branched structure" (Pew, 2003). Two, the advent of interactive programming gave the programmers lot more flexibility in coding and made their job less tedious and more productive. Programmers could now debug their code small sections at a time. The final years of this era saw the emergence of packet-switched networks, leading later on to the email revolution, spreadsheets, and games (Pew, 2003).

Context of SE and HCI: This age started, as part of the structured programming movement, the need for acquiring user requirements and understanding of "what the user would want to see and what the user wanted to type" (Pew, 2003). This age represents the beginning of the need for usability because of the direct use of computers by end users. This was the time that "led to much confusion and user frustration that was only occasionally addressed directly by usability studies". This era also saw the emergence of the human-factors engineers interest in computer systems, the introduction of concepts from cognitive psychology, and physiological methodology to design and evaluate human-computer systems (Pew, 2003). Work by Newell and Simon (1972) in the area of information processing theory of human problem solving spawned the now well-known techniques of think-aloud protocol taking. With this ground-breaking work a new partnership between Card, Moran, and Newell emerged which resulted in the birth of serious HCI research (1980; 1983). This period also saw the emergence of HCI guidelines and principles (Hanson, 1971). One can argue that this is possibly the start of the shift between the user interface and functional concerns of systems development because of the rather different backgrounds of the people involved in the functional side (traditional computer sciences) of things and those interested in the usability side (psychologists).

2.2.3 The GUI/OO years – 1978 to 1988

Target users of the era: Mostly trained personnel and a few computer enthusiasts

Players and visionaries: Donald Norman (behavioral science and human information processing to HCI, analytic framework to study human interacting with a computer, "Psychology of Everyday Things" (Norman, 1988))

Technology: This age saw the advances in hardware that finally paved the way to Graphical User Interfaces (GUIs) as we know them. Bit-mapped graphics made it possible to have selective redisplay of complex fonts, icons, and images, and memory cache systems supported multiple window displays.

Context of computing: The Xerox Star secretarial workstation in 1982, Apple Lisa in 1983, and Macintosh in 1984 revolutionized the computing arena. This era started the interaction paradigm of "What You See Is What You Get" (WYSIWYG). It was no longer necessary to wrestle with abstruse command-line formatting while interacting with word processors. Computing is now

truly within the reach of novice users. Englebart's mouse, the keyboard, Xerox Star's desktop metaphor, with Shneiderman's direct manipulation (1983) would become a standard of GUI paradigm until today.

Context of SE and HCI: On the SE side, this was the time of object-oriented (OO) programming and the realization that the waterfall model (Royce, 1970) is no longer effective (Pew, 2003). The OO approach facilitated the separation of user interface and functional core separation, marking the beginning of the separation of the two components of an interactive system: UI and backend. The limitations of the waterfall model prompted experimentation with more iterative development processes. On the UE side, there were increasing expectations for ease-of-use on the part of novice users. The user interfaces started to become complex and the interaction conceptually became an event-driven one where the user generated events, without any particular order, by accessing the UI widgets on the interface (Pyla, et al., 2004b). A direct consequence of this is the emergence of a new group of researchers attempting to study the theory, principles, and techniques about the UI component; transforming the field of HCI into a professional discipline (Pew, 2003). Therefore, this age can be considered the time that started the independent maturation of the SE and HCI domains. Even though two of the seminal attempts to bridge the UE and SE gap were made during this time (Draper and Norman, 1985; Mantei and Teorey, 1989), we argue that the disconnect between the two domains increased from then onwards for a significant number of years (see Section 2.3 for description of these attempts).

2.2.4 The Internet years – 1989 to 1999

Target users of the era: Wide variety of users from different age groups and with different skill levels.

Players and visionaries: Deborah Hix and Rex Hartson (provided a complete approach to engineer the usability process and guidelines for UI design); Jacob Nielsen (introduced discount usability); Ben Shneiderman

Technology: The TCP/IP protocol and packet-switched networking made a giant network of computers across the world called Internet possible. Email became ubiquitous and hypertext and hypermedia made sharing of a rich set of multimedia a reality (Pew, 2003). A variety of browsers

were developed to access this increasing amount of data that became available at a click by users.

Context of computing: During this period, the Internet became available to the common person and thereby revolutionized the way business is done across the world. This era also saw the emergence of Computer-Supported Cooperative Work (CSCW) where people from across the world could communicate and collaborate. Other dominant applications that came out of this era include person-to-person communication (instant messaging), File Transfer Protocol (FTP), video teleconferencing, and attachment support for email (Pew, 2003). With these technologies, geographic boundaries were quickly surmounted and the world truly became one step closer to the concept of a single global village.

Context of SE and HCI: This time saw the maturation of both SE and UE domains. Life cycle processes were becoming standardized and were beginning to be adopted by organizations across the world. Even though the success rate of software development projects never really achieved the same level as in other engineering domains, one could say that at least the scope of the software development problem had become more or less understood. The start of this era saw some of the first books on the process, activities, and issues involved in developing UIs (Mayhew, 1992; Hix and Hartson, 1993; Nielsen, 1993; Preece, et al., 1994). The need for connections between the SE and HCI were being recognized during this time. Hix and Hartson talk at length about the SE and UE life cycles and clearly show the parallels and the high-level relations between the two processes (Hix and Hartson, 1993). We argue that this era saw the beginnings of the realization and identification of the problem of gaps between the two life cycle processes.

2.2.5 The ubiquitous computing years and the start of the social networking era– 1999 to present

Target users of the era: Computing truly became available to people from all walks of life. People from almost all ages, cultures, professions began using computers.

Players and visionaries: Mark Weiser (introduced the concept of ubiquitous computing (Weiser, 1991) which would be the focus of this age), creators of social networking sites such as MySpace (2007c), Facebook (2007a), and Orkut (2007d).

Technology: These are the times of ubiquitous computing (Weiser, 1991; Weiser, 1993; Weiser, 1994). Computers were proliferated in different sizes and form factors. Mobile (wireless) technologies made it possible to be connected even when one is away from one's office. Handheld computing (using PDAs, cellular phones, etc) became hugely popular and quickly the users were inundated with a variety of computing options such as smart phones, wearable computing devices, PDAs, intelligent environments, portable audio systems, digital cameras, etc. This ubiquitous availability of computers and the pervasiveness of the Internet made it possible for vast numbers of people to virtually get connected with one another, forming social networks which in turn facilitate online collaboration and information sharing.

Context of computing: The face of computing has changed from the personal computer world of the previous two eras where the paradigm was of a single user interacting with a single computer. The proliferation of notebook computers and other mobile computing devices redefined this paradigm and continues to change our work environment, driven by the convenience of portable computing. As a consequence, users are forced to orchestrate a complex interaction between multiple devices, moving data and information back and forth, to accomplish their tasks. With the deployment of computing in various forms and factors, users are forced to wrestle with multiple technologies. Users trudge out USB key drives, remote desktop software, e-mail and network file storage in an attempt to orchestrate this complex interaction. The user interaction is no longer limited to the mouse and keyboard; pen-based, voice, gesture, 3-D interaction, etc. gained prominence quickly.

Context of SE and HCI: In spite of the commendable level of maturity achieved by these two disciplines independently, software systems with interactive components still fall far short of being perfect. And interactive-system development projects still suffered failures due to disconnected processes, unrealized functionality, poor usability, coming in late in schedule and over budget. This led to a growing level of awareness about the need to finally bridge these two life cycle processes.

Taking a historical perspective, it can be reasoned that one of the most important reasons why SE as a field has failed to connect to the UE discipline is that the processes and activities in SE have their roots in non-interactive, batch processing based, systems-side applications development

from the previous eras. On the other hand, one plausible explanation for why the UE discipline failed to connect to the SE domain is a lack of knowledge on the part of the usability engineers about the processes, constraints and schedules of SE. One can deduce that this lack of understanding about SE issues by the UE practitioners and vise versa is because of computer science curricula that attempt to teach students each subject exclusive of the other. Thus, in neither the SE nor the UE courses do students learn about the other domain and barriers involved in coordinating these two processes (Pyla, et al., 2004b). However, these years are starting to see an increasing amount of research towards bridging the two domains (2003c; 2003a; 2003b).

2.3 Bridging the Gaps between SE and UE

Over the years, a variety of efforts have been undertaken by many to bridge the gaps between SE and UE life cycle processes. One of the earliest attempts to relate the issues involved in the design and development of user interfaces to that of software engineering was by Draper and Norman (1985). This work predominantly tries to draw analogies between the software engineering way of thinking and the then new area of HCI. For example, using the software engineering metaphor to "run" a program on a particular platform or hardware, the authors suggest that a similar analogy where the user interface "can be thought of as being *run* on human users" and that the concept of a program "bug" should be modified "to allow for part of the system to be a person" and that there is a need for new "performance criteria for the combined human-plus-interface system." Another analogy the authors use is that of the speed-memory tradeoff in software programs: a program that is optimized for speed uses more memory to hold the necessary data-structures whereas a program optimized for memory use performs more operations to write and retrieve data, thereby loosing on speed. The authors point out that user interfaces too can be optimized for either ease of learning and use or for speed and power. Using such analogies, the Draper and Norman try to introduce the complexities and other issues arising with this new component of a software system called the user interface. They also point out the need for different specifications to account for the UI component, the need for separating the UI and backend, the need for different programming languages for developing UIs, and how input methods need to be standardized and abstracted to ease the programmer from being distracted with the details of different input devices. For the time it was written, this was an insightful and

visionary look at bridging the gaps between the SE and UE disciplines (Draper and Norman, 1985).

The later years saw more attempts to address this disconnect between the two life cycle processes. With an exception of one approach that merits mentioning separately, these efforts can be classified under the following main themes. Each of these themes will be briefly described in the following sub-sections.

2.3.1 Embedding one life cycle's techniques into another

The most popular approach to bridge the gaps between the SE and UE domains seems to involve embedding one life cycle's techniques into another or training one domain role with the skills required by the other role. For example, Ferre proposes integrating "usability activities" into the SE life cycle to *integrate* the two life cycles. He identifies the so-called "characteristics that define a user-centered process and choose the usability techniques and activities best suited for inclusion in the software [SE] process" (Ferre, 2003). This work identifies usability engineering activities that bear a high-level similarity to software engineering activities and merges them into the software engineering life cycle. For example, the UE's task analysis is mapped with "problem understanding" on the SE side. These usability activities were derived from the literature and were grouped into related categories.

Similar approaches include integrating the UE activities into the requirements engineering process (Ferre, 2003), into the Rational Unified Process (RUP) in SE (Sousa and Furtado, 2003), and into the UML specification or similar modeling languages of SE (Barbosa and de Paula, 2003; da Silva and Paton, 2003).

The problem with this approach is that the SE role must be trained in the UE activities and terminology. Even after such training, one cannot eliminate the inherent functional bias in the development team because of their primary area of expertise. Software development effort often involves trade-offs of many kinds between the two sides in a project (resources, level of functionality, extent of usability testing, etc.). This in turn involves "selling" of ideas for particular activities and negotiation for resources. Because of this, people in what is still essentially one role will be in a conflict of interest with needs of the other role. The SE role will usually opt to build in the functionality over doing usability testing, for example, if the schedule

is tight. Also, another problem with this approach of integrating UE activities into the SE life cycle is that usability engineers often adopt particular usability activities from a large set of possibilities based on the nature and context of the project. Arbitrarily selecting some usability activities that appear to be similar to SE activities runs the risk of using inappropriate or unsuitable set of techniques towards developing user interfaces because of the limited knowledge of the SE role about the UE process. Expecting one development role to learn the activities, techniques, or notations from another domain is not the best solution to bridge the gaps between SE and UE.

2.3.2 Architectures that support the needs of one process in the other

Another approach to bridge the SE and UE gap is the use of software architectures to ensure usability in the developed product. The rationale behind this approach is that the attributes of a software system are dependent on its underlying architecture (Bosch, 2000). In this approach usability is treated as a quality attribute that a software system should have (Bevan and Azuma, 1997; Bevan and Bogomolni, 2000). Juristo et al (2003) and Ferre et al's (2001) work is an example of this approach. This work considers usability as "just another quality attribute" which can be provided by an architecture like any other quality attribute. Juristo et al calls this approach a "forward engineering-perspective to usability in software architectures, as opposed to the conventional backward-engineering alternative of measuring usability on a finished system and improving it once the system is practically complete" (2003). We believe this argument is flawed, and shows a deep misunderstanding of the UE processes, on two counts: one, usability is not just a quality attribute that can be "plugged" into a system by adopting the "right" architecture. Two, UE is not a "backward-engineering" approach that tries to measure usability on a "finished" system. Neither is it true that the attempts to "improve" the usability of a system are undertaken after the system is "complete." In fact, when the UE process is properly applied, formative usability evaluation is a "forward-engineering" method for developing an evolving interaction design.

Another well cited work that uses the architectural approach to usability is by Bass and John (2001; 2003). They consider usability an "important attribute" of the system that should be considered "during all phases of software design, but especially during architectural design

because of the expense involved in adding usability aspects after users have tested the system". The authors point out that the traditional architectures such as the Model View Controller (MVC) (Krasner and Pope, 1988) have been useful in separating the user interface from the functional core and localizing the effects of frequent changes that are usual on the user interface side. They argue that these architectures that provide the separation of concerns between what we call SE and UE efforts are by themselves not sufficient. This is because certain usability aspects transcend the user interface code and have implications at the functional level. The authors give the example of a "cancel" usability aspect and how it requires an architecture that allows command processes to record initial state, supports a separate thread to listen for this command, and contains a mechanism to inform all the impacted parts of the overall system. The authors have identified 26 usability features such as cancel and provide architectural implications to each of those aspects. The authors do not mention a development process or how the use these architectures map into a SE or UE life cycle.

Ensuring the usability of a system requires a detailed and complex process that is frequently punctuated by different types of evaluations. In the big scheme of things, we acknowledge that software architectures do play a significant role in ensuring that the developed systems are usable, by providing support for usability features in the functional core. However, without a defined process they are not sufficient by themselves to ensure usability.

2.3.3 Customizable life cycles and frameworks

The last few years have seen the emergence of software development methodologies that are less process, documentation, and planning intensive and more agile, development-team-tacit-knowledge based, and styled for "embracing change" (Beck, 1999; Beck, 2000; Horrian, Mahmud and Karthikeyan, 2003; Paetsch, Eberlein and Maurer, 2003). These new paradigms are called Agile Software Development Methodologies (Ambler, 2002; Ambler, 2004), methodologies which have created a heated debate among practitioners and researchers. While the proponents of agile methods strongly argue the merits of these methods, opponents usually prefer the plan-based traditional approaches (Deursen, 2001; Duncan, 2001; Glass, 2001; Boehm, 2002; DeMarco and Boehm, 2002; Leonardi and Leite, 2002; Kääriäinen, et al., 2003). However, almost all the software development methodologies are points on a continuum of a planning spectrum where the extremely unplanned approaches by hackers fall on one extreme

and “inch-pebble ironbound contracts” (Boehm, 2002) lie on the other end. Every other software development methodology falls in between with agile methods being closer to the hacking end and milestone plan-driven models towards the other extreme (Boehm, 2002).

Extreme Programming (XP) is one of the relatively new software development methodologies that have the flavor of a traditional UE life cycle. In the words of XP’s creator Kent Beck, “XP is a light-weight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements.” XP identifies that risk is the basic problem and cost, time, quality, and scope to be the four variables in most software development efforts. XP also has four values: communication, simplicity, feedback, and courage. XP advocates “many practices that can’t be done without communicating” such as unit testing, pair programming, and task estimation. Simplicity is stressed in the sense that the simplest possible solution to a problem is recommended for adoption. Frequent unit testing is used as a feedback mechanism to monitor the state of the system. XP advocates the courage to be able to make fundamental changes in the architecture of the system if necessary and to throw away code and start over if needed. Therefore, the basic principles of XP include rapid feedback, assuming simplicity, incremental change, embracing change, and quality work (Beck, 2000).

Philosophically, this is not different from most of the UE development processes, but there are large differences in the details. Compared to the heavy weight and rigid SE processes with little feedback and evaluation components of the past (such as waterfall model (Royce, 1970)), XP advocates iterative development, frequent evaluation, evolutionary approach to development, user stories (requirements in XP are in the form of stories written by actual users), metaphors (XP calls the system architecture a metaphor), etc. XP can be seen as a kind of UE life cycle process applied to SE development. Interestingly, XP does not mention any techniques or activities for UI development in an XP-based development methodology.

Even though agile methods (such as XP) do not explicitly mention user interface development processes, we believe that the underlying philosophy of these methodologies to be flexible, ready for change, and evaluation centered has the potential to bridge the gap between SE and UE if they are extended to include the UI development components and techniques.

2.3.4 CS education and curricula standards

Traditionally, the concepts, theory and techniques associated with the user interface (UI) design domain are taught as a part of Human Computer Interaction (HCI) courses in most computer science departments. These courses use text books such as (Nielsen, 1993; Preece, et al., 1994; Newman and Lamming, 1995; Shneiderman, 1998; Rosson and Carroll, 2002) and cover topics that are completely focused on the user interface side of system development. Little or no information is provided on the software architectural implications that arise from user interface design decisions. For example, Rosson and Carroll talk about scenarios and focus on the UE life cycle and process (2002). They do not provide a clear description of the two development efforts and the implications for the functional core. The incremental case study provided in this book proceeds through different interface design stages (as the chapters progress), and ends up with a system entirely developed and including user documentation. The readers are not informed how the user interface specifications are communicated to the software developers, nor how the design constraints are negotiated between the two development bodies.

Students who complete this type of academic HCI course gain little, if any appreciation for the development process associated with the functional part of the system. They may wrongfully believe that any usability or user interface specification derived will "somehow" get automatically implemented by "someone", which in reality is rarely the case. To the contrary, simple usability features, such as undo, that the students might take for granted, can have serious architectural implications in the functional part of the system (Bass and John, 2001).

The SE courses as they are offered in most computer science departments are no better than their HCI counterparts when it comes to addressing the connections between the two domains. SE courses often omit any references to user interaction development techniques. We surveyed nine out of 13 of the top selling books in software engineering/software design and engineering (according to facultyonline.com website) (2007b). Out of the nine books, only three identify and dedicate sections to user interface development. Even so, those three still do not address the dependencies and implications of the two processes on one another. Students completing software engineering courses that follow the syllabus in textbooks like these obviously will gain little or no appreciation for the user interface issues and the constraints on the functional core that may arise due to them.

Some textbooks have attempted to address this issue, but their impact in changing the academic environment has been minimal. For example, Hix and Hartson (1993) discuss at length the connections that should exist between UE and the rest of the SE lifecycle. However, because their book focuses on usability engineering methods and techniques and does not suggest how to integrate the two domains, the SE community is not really aware of its impact.

In recent years there has been a growing awareness of the importance of bridging the gap between the SE and UE domains (Pyla, et al., 2004a; Pyla, et al., 2005). There have also been appeals for curricula that integrate these two disciplines (Douglas, et al., 2002; McCauley, 2003; Pyla, et al., 2004b). For example, Sefah (2003) points out that there are very few software engineers who understand the human-centered design process. He states that one of the reasons for this is the lack of a proper educational framework. He also provides a list of skills that one should have to perform human-centered design. Latzina and Rummel (2003) reason that because of the lack of HCI studies available for computer science students, much of the usability training is left to corporate training workshops. Because of cost and time factors, these workshops are extremely short (about 2 days) in duration, thereby reducing the course content to "commonplace statements". Wahl proposes to teach SE students "usability testing" so that "students can learn about user-centered design and what makes software usable by running usability tests" (2000). We believe this focus on usability testing alone to be unsuitably limited as it promotes the idea that UE is just usability testing that is performed at the end of software development, and that this is enough to teach user-centered design.

Leventhal and Barnes have been advocating and implementing a curriculum that integrates HCI and SE within a computer science course that "emphasize(s) some SE notions in the context of HCI concepts" (2001; 2003). They incorporate some of the SE topics into a project oriented HCI course. On the other hand, Veer and Vliet appeal for a "minimal" HCI course to be incorporated into a software curriculum (2001) to train students for a more integrated approach towards development of interactive systems. Similarly, the joint task force on computing curricula commissioned by ACM Education Board, IEEE-Computer Society Educational Activities Board, and other professional societies, also recommends bits and pieces of HCI in their SE courses (The Joint Task Force on Computing Curricula, 2004). They recommend a separate HCI course (SE212), similar to a number of pure HCI courses taught in universities, that covers the usability

engineering processes, methodologies, architectures and techniques. Unfortunately, their recommendations for core software engineering courses such as "Software process and management" (SE324), "Software Project Management" (SE323), "Software design and architecture" (SE311), etc. do not even mention user interface issues.

In our work at Virginia Tech, we have also addressed the education aspects of bridging the gaps between SE and UE (Pyla, et al., 2004b). We prescribe a balanced curriculum where the SE and UE life cycles are covered equally without any bias toward either of the processes or by incorporating pieces of one lifecycle's methodologies into the other. Leventhal and Barnes' (2001; 2003) suggestions are the closest to our own arguments, even though their approach is more focused on the HCI components of the curriculum. Moreover, Leventhal and Barnes do not address the issue of dependencies and constraints between the two lifecycles. We consider these to be one of the most important aspects the developers of tomorrow should comprehend.

2.3.5 Standards on life cycle processes

A variety of standards have tried to address the gaps between the SE and UE life cycles at one level or the other. Unfortunately, the significance of the UE life cycle and the importance of communication, coordination, synchronization, constraints and dependencies, and anticipation and reaction to change issues are not described or prescribed in most of the software development standards that exist today. For example, the 31-page IEEE-830 standard (1998) on recommended practices for software requirements specification (SRS) contains only about 10 lines(!) relating to user interfaces (Section 5.2.1.2 in the standard), and states that user interface specifications should be a part of the SRS. This part of the standard takes an ad hoc stab at a few user interface issues (e.g. required screen formats, page and window layouts, screen content, availability of programmable function keys, etc.) which seem arbitrarily chosen from the enormous possibilities not mentioned. More importantly, it says nothing about the UE life cycle process for creating the interaction design, which is a main part of the user interface software specification. It is misguided (and worse, misguiding) to expect the user interface specifications to be available that early in the requirements process without having followed a proper UE design life cycle. We believe that this document should have a reference to another standard for user interface interaction design requirements.

Another source of confusion with the IEEE-830 standard is that the items mentioned in this document such as required screen formats, page and window layouts, and screen content are design specifications for software engineers (the standard includes nothing about how to design them for usability). For the UE role, "requirements" are mostly stated in terms of usability attributes such as learnability, subjective satisfaction, ease of use, etc. Even these usability specifications are subject to calibration and quantification in later stages of the UE development process.

However, we do not disagree with the intent behind the idea that user interface requirement specifications for user interface software are properly a part of the SRS. But in reality it is not possible to generate requirements specifications for user interface software without going though an iterative process of interaction design and evaluation, but standards such as the above described IEEE-830 (on SRS) and IEEE/EIA-12207.1 (on software life cycle processes-life cycle data) (1997) do not acknowledge the kind of life cycle process that is needed to develop a high usability interaction design. Neither do they acknowledge the myriad relations and dependencies between the activities and work products of the SE life cycle with that of UE and vice versa.

2.3.6 Scenarios and use cases as bridges

Scenarios are narratives and envisioned design solutions about users' tasks as they are supported by the system. Scenarios are usually written in natural language and therefore afford a fluid communication between various roles that are associated with any software development effort. In the words of Rosson and Carroll (1995), "the shared context provided by the scenarios promotes rapid feedback between usage and software concerns, so mutual constraints and opportunities can be recognized and addressed early and continuously in the development process". These scenarios are then refined to determine objects, descriptions, and the interactions between various objects (this work is targeted towards the OO design paradigm). Software tools can be used to facilitate this transformation of scenarios (Rosson, 1999). Similarly, attempts have been made to use scenarios-based design for developing requirements of a software system (Carroll, et al., 1998).

Another of the task-narrative-based approaches is the application of use cases to connect SE and UE life cycles. A use case identifies "a thread of usage for the system to be constructed (and)

provide(s) a description of how the system will be used" (Pressman, 2001). Use cases are used in software engineering to specify requirements and to aid in modeling the design of the system. Use case generation does not usually involve end-users and is expressed in a system-centered notation. Some attempts to bridge the gaps between SE and UE use modified versions of use cases that are more amicable for communication with end users and which are expressed in more user-centered language (Constantine, 1995; Constantine and Lockwood, 1999; Alsumait, Seffah and Radhakrishnan, 2002). For example, Alsumait, Seffah, and Radhakrishnan (2002) use "Use Case Maps" to integrate "task analysis and usability requirement(s) into the traditional software requirement engineering process." Use case map "is a visual notation" that describe the starting points in an interaction sequence (with pre-conditions or triggers that cause this start of the sequence), "causal chains of responsibilities" which show the actions, tasks, or functions that should be performed, and end points with post-conditions or results of the task sequence. The idea is that such a narrative aids in uncovering interaction problems upfront during the requirements modeling phase itself (Alsumait, Seffah and Radhakrishnan, 2002). Other examples of modifying or supplementing the traditional software engineering use cases to account for usability aspects include "essential use cases" (Constantine, 1995; Constantine and Lockwood, 1999)

Even though we recognize the utility of this kind of an approach, we believe that these techniques require skilled personnel who are well versed with the HCI and UML/Use Case aspects of modeling software systems. In reality, it is difficult to have designers proficient in both the SE and UE techniques and notations. We acknowledge that these narrative based approaches afford a high level of communication between the two development roles from SE and UE. However, we believe that for such techniques to be effective there is a need for a more independent process-oriented approach to building systems with interactive components.

2.3.7 A common framework approach

The one related work that does not fall under any of the previous categories but merits discussion is by Pawar (2004). This work was undertaken as a part of a Masters Thesis in Computer Science here at Virginia Tech in close collaboration with us. Pawar proposes a "common framework" that is based on the definition of an "interface between the UE and SE processes." This work recognizes the differences in focus, terminology, and techniques between the SE and UE

processes, and defines some coordination and synchronization points where the two life cycle roles come together. This work also identifies what information should be exchanged at these synchronization points. This work differs with ours in the following fundamental ways: one, while Pawar's approach focuses on particular life cycle processes for the SE and UE, our work does not align itself to a particular development methodology. Pawar uses the Scenario-Based Design approach to UE and the Requirements Generation Model combined with Structured Analysis and Design for the SE life cycle. Our approach is general and can be applied to any development methodology in either domain. Two, by specifying static information exchange points, Pawar's approach does not account for the different levels of iterativeness in the two life cycle processes. We account for this by using Ripple's messages (Section 4.2.3) and not specifying static information exchange points.

2.4 Summary

Recent years have seen a significant increase in the amount of research towards bridging the gaps between the SE and UE disciplines (2003a; 2003b; 2003c). Starting from the early efforts to couch the UE concerns in SE terminology (Draper and Norman, 1985) there have been a wide variety of approaches taken towards connecting the SE and UE life cycles processes. In our literature review, we found most of these efforts to fall under six main categories: embedding one life cycle's techniques into another (Section 2.3.1), using architectures that support one process's needs in the other (Section 2.3.2), using customizable and flexible life cycle processes and frameworks (Section 2.3.3), using cross-pollinated CS education curricula (Section 2.3.4), specifying standards on life cycle processes to include techniques from the other domain (Section 2.3.5), and by employing scenarios and use cases as bridges (Section 2.3.6).

3 Chapter Three: Ripple Description Model

3.1 Introduction and Background

Systems development is a complex and dynamically changing endeavor with a highly interconnected set of activities, each performed by one or more developers. In order to understand the interactions among these entities in the development space, we first need to formally define and describe these entities. Only with a complete and unambiguous description of the various entities can we *specify* the relationships among them, and the consequences of realizing those relationships. Formal specification languages have been used to describe relationships among entities in different domains. For example, in the physical world "objects move, collide, flow, bend, heat up, cool down, stretch, compress, and boil" and "to understand commonsense physical reasoning" and to "make programs that interact with the physical world as well as people" one needs to understand "when they occur, their effects, and when they […] stop" (Forbus, 1985). The Qualitative Process Theory (Forbus, 1985) defines a formal language notation that can be used to specify dynamic behavior and physics of everyday physical objects and how they interact with one another in the world. Similarly, formal notations have been used in a manufacturing context to describe knowledge about different entities such as materials and chemical properties of graphite-epoxy composites to manage closed loop control systems (Matejka and Lagnese, 1998).

In the software development domain, formal modeling has been used to represent file-level relationships in a software development environment to describe and facilitate concurrent access to these files by different developers using a software development environment (Barghouti, 1992). Similarly there are other formal specification languages such as ISO LOTOS that describe the structure of a problem with the idea that it can later be translated into executing software code (1989). Other process specification languages such as Visual Process Language (VPL) attempt to describe the software development process as a workflow with start and finish as endpoints and with a series of "enactions" in between (Shepard, Sibbald and Wortley, 1992). However these process specification languages are targeted mainly towards the SE life cycle alone and do not focus on the relationships among different entities which are critical in an interactive-software development space.

Also, another key aspect that these process description languages do not emphasize is communication. For an interactive software system, as development proceeds in each life cycle process, new activities are started, work products are created, new insights are gained, and these insights, in turn, require changes/updates to work products generated in previously completed activities. The key to ensuring that these changes and updates are considered by the two roles is dependent on *communication* of these needs at just the right time and with just the right developers. In this chapter we describe the Ripple Description Model which expresses the various entities and relationships within the systems development space in order to facilitate this communication.

3.2 RDM

In the context of interactive-software development, the Ripple Description Model, RDM, is expressed as:

$RDM = <L, M>$, where

L is a process description language used to describe system development life cycle instances, and

M is a set of mappings among work activities and other elements of a life cycle instance that embodies a representation of the communication needs.

3.3 Life cycle process description language

The first major component of the Ripple Description Model is the life cycle process description language, L, expressed as:

$L = < V, G>$, where

V is the vocabulary of the language, and

G is the grammar of the language.

3.3.1 Language grammar and vocabulary

V and G determine the set of well-formed expressions in L. In practice, these well-formed expressions are used to produce descriptions of life cycle instances within development projects.

Henceforth we will use the term "life cycle" to refer to a life cycle instance except where necessary to disambiguate.

Within L, the vocabulary V consists of "terminals" denoted by V_T, and "nonterminals" denoted by V_N. The terminals are a set of the lowest level life cycle work activities, including their names and other attributes. The nonterminals are a set of "blocks" or structured groups (as an abstraction) of work activities. Thus

$$V = V_T \cup V_N$$

The grammar G describes life cycle structure in terms of its grouping, sequencing, iteration, and conditionals. Figure 6 is a schematic diagram of a typical UE life cycle with its V_T and V_N and a graphical example of sequence and grouping that can be expressed in G.

3.3.2 Work activities and work products

A work activity is the fundamental unit of a system development life cycle. Each $wa_i \in WA$ has the form:

wa_i = <*wa-symbolic-name, LC-type, wa-type, wa-technique, wp-affected, developer-role, developer-name*>, where

i is the unique (internal) identifier of the work activity instance. In practice within expressions in *L*, the *wa-symbolic-name* is a mnemonic device to identify the corresponding wa_i in a life cycle description. For example, a work activity instance expression might be *cognitive-walkthrough*. If two work activities, wa_i and wa_j, have identical elements in their tuple, to maintain their mnemonic value, they should be distinguished by assigning distinctive *wa-symbolic-names*, for example, *cognitive walkthrough-1* and *cognitive walkthrough-2*.

LC-type ∈ {SE, UE} is the type of life cycle (software engineering or usability engineering) with which this *wa* is associated;

The element *wa-type* is the name of the kind of development activity such as task analysis in UE and verification and validation in SE.

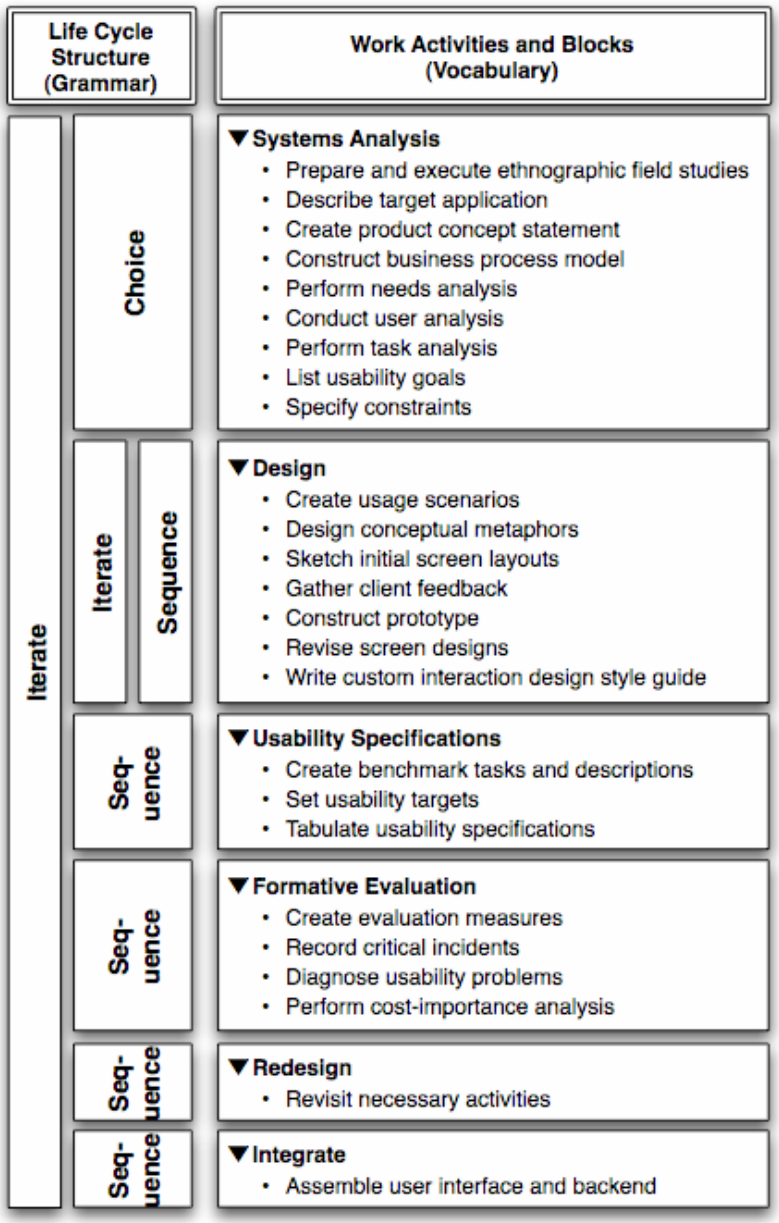


Figure 6: Work activities in a typical UE life cycle

The element *wa-technique* is the technique to be used to perform the work activity, such as expert inspection or lab-based testing, either of which could be used to perform the formative evaluation work activity type in UE.

The element *wp-affected* is the work product artifact, *wp*, that is created as the result of a work activity, wa_i. For example, a hierarchical task analysis diagram might be created as a result of a task analysis work activity type in a UE life cycle, usability problem lists may result from

formative evaluation in UE, and software requirements specifications can result from requirements analysis in SE.

The element *developer-role* is the life cycle role of the person who is in charge of performing the work activity. Most life cycles have specialized developer roles for a specific work activity, such as usability evaluator for formative evaluation, software coder for software implementation, and software tester for code testing. Sometimes there are multiple experts performing a single developer role for a given work activity. Often a given individual is capable of performing multiple roles.

The element *developer-name* is the name of a specific individual who, in the developer role for a particular work activity, is in charge of performing that work activity. This element uniquely identifies a particular individual from a set of team members who could take on the developer role for a given work activity.

These elements of a work activity tuple allow a complete description of a given work activity instance. For example, a work activity instance named *CW-1* and assigned the unique identifier wa_1, representing a formative usability evaluation conducted by a usability evaluator named John Doe is:

wa_1 = <*CW-1, UE, formative usability evaluation, cognitive walkthrough, usability problem list, usability evaluator, John Doe*>

Henceforth, we will use the term "work activity" to refer to a work activity instance except where necessary to disambiguate.

3.3.3 Blocks of work activities

3.3.3.1 The need for blocks of work activities

Figure 6 shows the need to group work activities into "blocks," each of which can be named as an abstraction of the contained work activities. For example, "Design" is a high-level descriptor of a block of work activities including create usage scenarios, design conceptual metaphors, sketch initial screen layouts, construct prototypes, etc. In L, a block is one or more vocabulary elements (of V_T and/or V_N) combined by elements of V_N according to the rules and productions of

the grammar, *G*. In practice the kinds of blocks described in *G* reflect the real-world reasons for aggregating work activities, including (for example) *iterate*, *choice_of_any*, *ordered_choice_of_any*, *choice_of_one*, *do_sequence*, etc., which are reflected in the terms V_N:

$$V_N = \{iterate, choice\_of\_any, ordered\_choice\_of\_any, choice\_of\_one, do\_sequence, \ldots\}$$

The ellipsis indicates open-endedness, allowing users of *L* to define their own block types (elements of V_N) as needed.

We define each block within a life cycle description as a kind of abstract work activity which has the form:

$block_i$ = <*block-symbolic-name, LC-type, block-activity-type, block-structure-type, set-of-contained-work-activities, block-definition, block-developer-name, pma*>, where

The element *block-symbolic-name* is a mnemonic device to identify the corresponding $block_i$ in a life cycle description. For example, the first block coming from the diagram in Figure 6 might be called (assigned *block-symbolic-name*) *Analysis*.

LC-type $\in$ {SE, UE} is the type of life cycle (software engineering or usability engineering) with which this block is associated.

The element *block-activity-type* is the name of the high-level development activity type such as systems analysis in UE and integration and test in SE.

The element *block-structure-type* identifies the kind of block structure represented. For example, the block named *Systems Analysis* (from Figure 6) would be a *choice_of_any* structure.

The element *set-of-contained-work-activities* is the set of all $wa_i \in V_T$ named within the block definition. This set can be used to determine aggregate attributes from the constituent work activities by performing set theoretic operation on this *set-of-contained-work-activities*. For example, one could compute the set of all work products affected within a block or the set of all developers involved in the activities within a block.

The element *block-definition* is the definition of the block in *G*, in terms of work activities (elements of V_T) and aggregating terms in V_N.

The element *block-developer-name* is the name of a specific individual who has taken the lead role for the work activities contained within a block. This element uniquely identifies a particular individual from a set of people who have the authority and responsibility to perform what is required in the particular block. This lead person would be the point of contact by other roles for this abstract work activity. For example, a block instance named Analysis and assigned the unique identifier $block_1$ representing the systems analysis phase lead by Jane Doe is:

$block_1$ = <*Analysis, UE, systems analysis, choice_of_any, {work activities in block}, BL1-definition, Jane Doe, pma*>, where

the definition of block, could include work activities such as prepare and execute ethnographic field studies, describe target application, create product concept statement, construct business process model, perform needs analysis, conduct user analysis, perform task analysis, list usability goals, and specify constraints.

3.3.3.2 Grouping into blocks

The full details of *G* and how it is used to define blocks are not given here, but are well known within the computer science discipline. To represent the concept of grouping work activities into higher-level blocks for the purpose of abstraction in life cycle descriptions, an example of how the grammar could be defined is as follows:

block → *block-symbolic-name: BEGIN block-activity-type*

block-contents-list

END block-activity-type' where

block-activity-type = *block-activity-type'*,

block-contents-list → *block-contents-list-item* | *block-contents-list-item, block-contents-list-item*, and

block-contents-list-item → $v_T \in V_T$ | *block*

For example, the *block-definition* of the *Systems Analysis* block of Figure 6 might appear in *L* as:

block → *Analysis: BEGIN CHOICE_OF_ANY*
Prepare and execute ethnographic field studies
Describe target application
Create product concept statement
Construct business process model
Perform needs analysis
Conduct user analysis
Perform task analysis
List usability goals
Specify constraints
END CHOICE_OF_ANY

Where *Analysis* is the *block-symbolic-name*, *CHOICE_OF_ANY* is the *block-structure-type*, and the list of work activities between the *CHOICE_OF_ANY* tags is the *set-of-contained-work-activities*.

3.3.3.3 Project management attributes of blocks

Since project managers use a development life cycle instance to manage a project, the life cycle representation must also contain some project management attributes, such as schedule (planned completion dates) and budget (resources). As an example, for the previous UE life cycle shown in Figure 6, we can add some project management attributes to the life cycle description to represent the planned completion date as shown in final column of the life cycle schematic diagram in Figure 7.

The final tuple element of the definition of "$block_i$" in Section 3.3.3.1 is "pma", which represents project management attributes of the block. A project management attribute is:

pma=<schedule-attributes, budget-attributes, personnel-attributes, space-attributes, other-constraint-attributes, other-resource-attributes>

These attributes are used by project managers to associate resources and constraints with each block of work activities.

schedule-attributes include such parameters (for example) as *planned-start-date*, *actual-start-date*, *planned-completion-date*, and *actual-completion-date* ∈ Calendar, which specify, respectively, the date the work activity was scheduled to begin, the actual start date, the scheduled date for completing the work activity, and the actual date the work activity was

completed. The list of schedule attributes would be tailored to fit any given project, and can be updated over time.

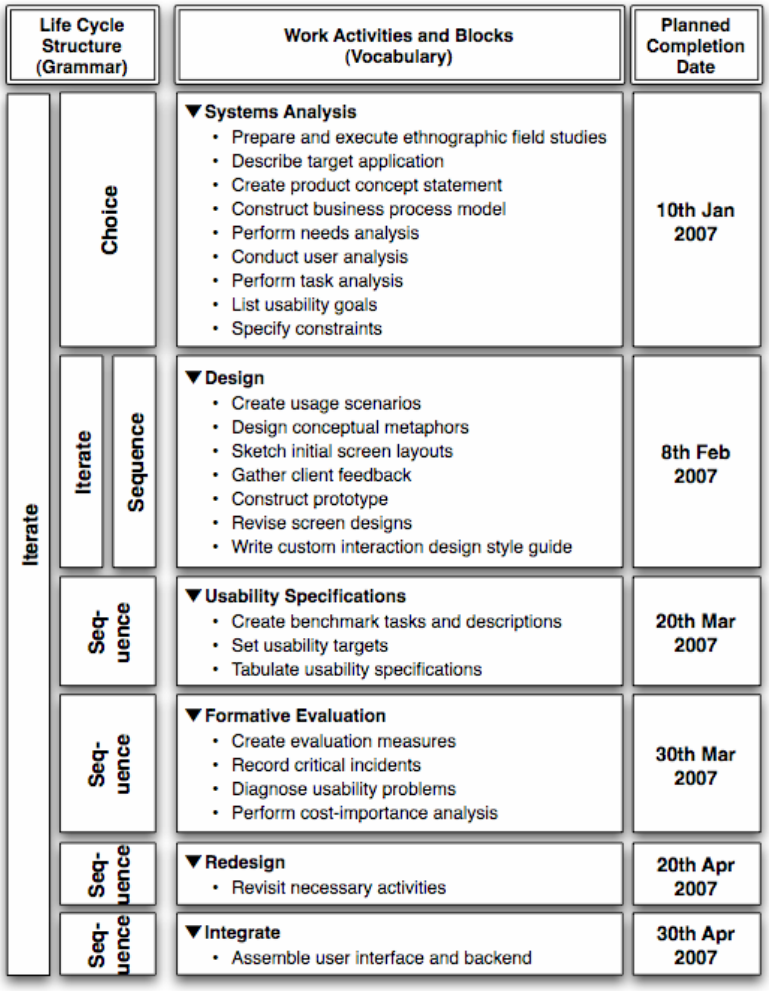


Figure 7: A typical UE life cycle with scheduled dates

The rest of the *pma* elements are more or less self explanatory. The other resource-attributes can include organizational assets such as computers, servers, personnel, etc. These elements of a project management attribute tuple allow a rather complete description of the constraints and resources associated with a block of work activities. For example, a project management attribute with assigned unique identifier pma_1 for a block (abstract work activity) named *Analysis* with a start (planned and actual) date of 13 July '06, planned completion date of 18 August '06, assigned personnel including John Doe, Jane Doe, and Little Doe, and allocated office of Room 153, would be:

pma_1 = < (07/13/2006, 1/10/2007, -), $10,000, ("John Doe", "Jane Doe", "Little Doe"), "Room 153", ...>

It is sometimes necessary to isolate particular dimensions of a *pma* in the context of project management and their associations with work activities or abstract work activities. For example, in the final column of Figure 7, we showed the *planned-completion-date* dimension of the *pma* tuples for each of the abstract work activities and excluded the other attributes such as *start-date* or *RCS* dimensions. This can be achieved using projections of a *pma*, as shown in the next section.

3.3.4 Projection functions

A work activity or block tuple describes multiple attributes of the activity or group of activities, each attribute spanning a different dimension in the development space. In other words, a work activity tuple can be considered to be a vector of dimensions such as work activity type, developer role involved, and work products generated. However, it is sometimes necessary to be able to look at one or more dimensions of a tuple representing a work activity or block (or any other tuple) but not the whole tuple. This requires a *projection* of the tuple.

More generally, if $x \in X =< x_1, x_2, x_3, ..., x_n >$, then x is a vector of n dimensions (dimensions 1, 2, 3, …, n) and x is a single point in that n-dimensional space having a value of x_1 in dimension 1, x_2 in dimension 2, etc. For example, consider a work activity:

wa_1 = <cognitive walkthrough, UE, formative usability evaluation, cognitive walkthrough, usability problem list, usability evaluator, John Doe>

In this case dimension 2, for example, is the *LC-type,* and the counterpart of x_2 is the value of wa_1 in this dimension, or UE.

A projection function, $\prod$, is used to extract a projection, an m-tuple, of an n-tuple, where $m \leq n$. In essence, $\prod$ simply extracts the desired elements from the n-tuple, the others being lost. For x shown above,

$\prod_i(x) = <x_i>$, where

$1 \leq i \leq n$; and

$\prod_{i,j,...,m}(x) = <x_i, x_j, ..., x_m>$, where

$1 \leq i \leq n$;

$1 \leq j \leq n$;

…

$1 \leq m \leq n$;

$i \neq j \neq ... \neq m$.

An example of a projection function of our work activity, wa_1, with respect to the *wa-type* and *wp-affected* for a formative evaluation, could show the outcome of such an activity to be a usability problem list:

$\prod_{wa\text{-}type,\ wp\text{-}affected}(wa_1) = <formative\ usability\ evaluation,\ usability\ problem\ list>$

Similarly, a projection function of a different work activity, wa_2, with respect to the *wa-type* and *developer-name* for a task analysis, could identify who is performing it:

$\prod_{wa\text{-}type,\ developer\text{-}name}(wa_2) = <task\ analysis,\ John\ Doe>$

3.3.5 Selection/filtering of work activities

The tuple used to define a work activity, $wa \in WA$, yields a fine-grained specification of a single work activity instance. Sometimes it is necessary to refer to subsets of *WA* with more coarsely-grained specification. For example, we may wish to refer to all work activities associated with a specific work product, a usability problem list, regardless of, say, the name of the developer who performed the work activity or the work activity type that was used to produce the list. This requires a selection of all the work activities whose *wp-affected* projection is a usability problem list:

$Y = \{wa \mid \prod_{wp\text{-}affected}(wa) = usability\ problem\ list\}$, where

Y is a set of work activities. A similar selection function can also be performed on block tuples.

3.4 The Mappings

The second major component of the Ripple Description Model is the set of mappings that are used to represent relationships that imply communication needs among work activities and

abstract work activities between and within the two life cycles. In the following sections we present the concepts necessary to describe such mappings.

3.4.1 Boolean state variables

A Boolean state variable is a Boolean function that takes on a value of true or false over time. In the Ripple Description Model, we define Boolean state variables that are dependent on the *state* of various projections of work activities and other development workflow parameters, such as work products, calendar events, and developer initiatives. States of these workflow parameters include such project-oriented conditions as "wa being performed", "wp being modified", "specific point on calendar reached", and "insight gained by developer". As an example, the primary Boolean state variable function that applies to a work activity is *is_being_performed*.

Figure 8 shows examples of this Boolean state variable for work activities wa_1 and wa_2 as they go through a state change from false ("activity not being performed") to true ("activity is being performed") and then back to false. Figure 8 also shows that work on wa_1 was revisited.

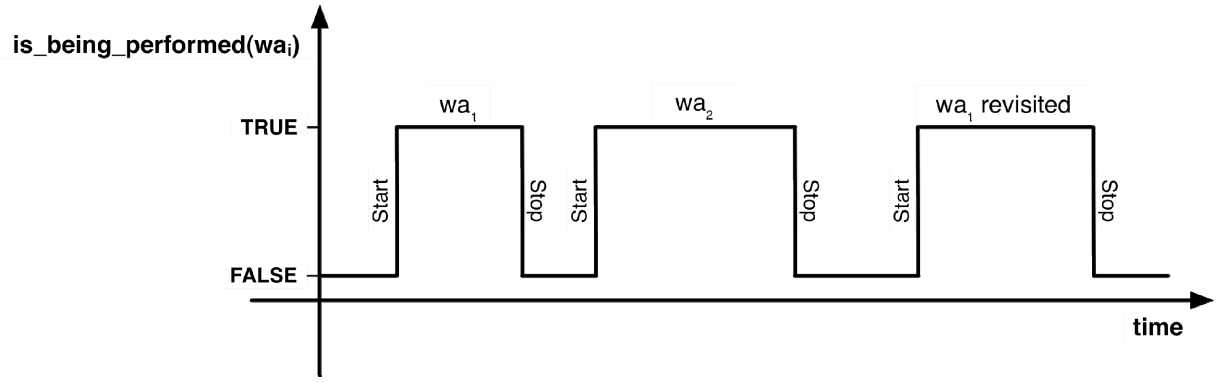


Figure 8: Boolean state values for work activities

The Boolean state variable *is_being_performed* is defined as:

$$is\_being\_performed(\mathrm{wa}_1) = \begin{cases} True, \text{ if and only if } \mathrm{wa}_1 \text{ is currently} \\ \qquad \text{being actively performed} \\ False, \text{ otherwise} \end{cases}$$

The primary Boolean state variable function that applies to a work product is *is_being_modified*. Figure 9 shows the Boolean state variables for work products wp_1, wp_2, and wp_3 as they go through state changes false ("not being modified") to true ("being modified") and back.

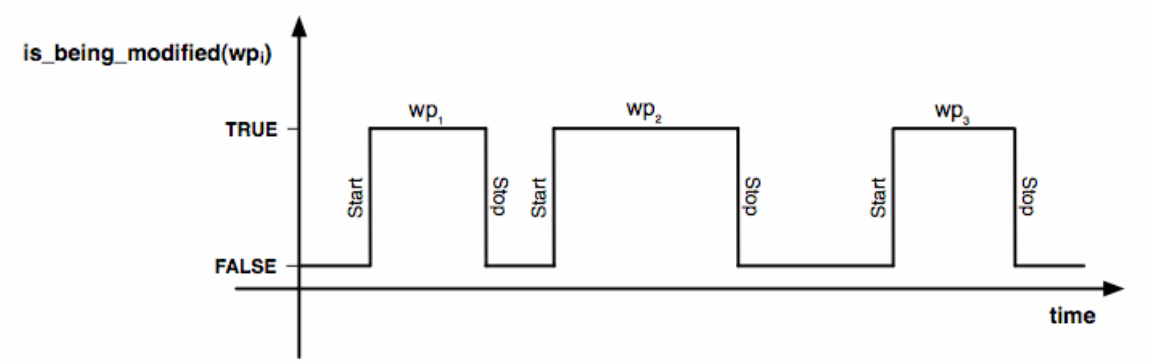


Figure 9: Boolean state values for work products

Similarly, suppose $\prod_{wp\text{-}affected}(wa_1)$ = <*usability problem list*>. The corresponding Boolean state variable *is_being_modified* is:

$$is\_being\_modified(\prod_{wp\text{-}affected}(\mathrm{wa}_1)) = \begin{cases} True, \text{ if and only if the problem list for } \mathrm{wa}_1 \\ \qquad \text{is being created, updated, or changed} \\ False, \text{ otherwise} \end{cases}$$

As an example, Figure 10 shows the Boolean state variables for developer insights. This describes the condition where someone in a particular developer role gains an insight into the project that might have an impact on other work activities in the project.

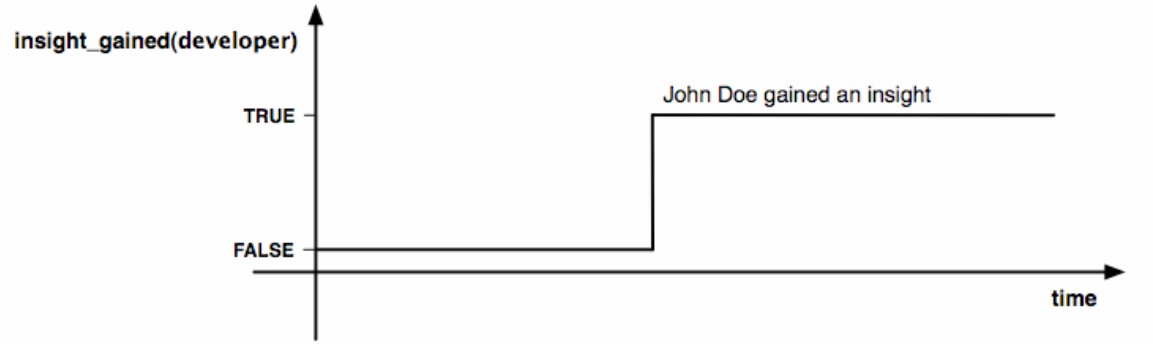


Figure 10: Boolean state value for developer insights

These types of unary Boolean state variables (a function of one parameter) can be generalized to n-ary Boolean state variables, as functions of multi-dimensional projections of a work activity.

Therefore if

wa_1 = <*cognitive walkthrough, UE, formative usability evaluation, cognitive walkthrough, usability problem list, usability evaluator, John Doe*>, then

$\prod_{\text{wa-type, developer-name}}(wa_1)$ = <*formative usability evaluation, John Doe*>, which can have an associated Boolean state variable

$$is\_being\_performed(\prod_{wa-type,developer-name}(wa_1)) = \begin{cases} True, \text{ if and only if John Doe is actively performing formative usability evaluation within } wa_1 \\ False, \text{ otherwise} \end{cases}$$

As another example, a Boolean state variable can be expressed as the following projection of a work activity:

$\prod_{\text{developer-role, developer-name}}(wa_1)$ = <*usability evaluator, John Doe*>

$$insight\_gained(\prod_{\text{developer-role, developer-name}}(wa_1)) = \begin{cases} True, \text{ if and only if John Doe, during the usability evaluation performed in } wa_1, \text{ gains an insight that might impact other work activities and developer roles} \\ False, \text{ otherwise} \end{cases}$$

Within a tool-based system development environment it is possible for the values of Boolean state variables such as *is_being*_performed or *is_being_modified* to be detected automatically. However, in practice, a Boolean state variable like *insight_gained* would necessarily have to be self-declared by the developer but could be valuable in communicating about an on-going development process.

Figure 11 shows another kind of Boolean state variable, which can be expressed as the following projection of a project management attribute:

$$date\_has\_arrived(\prod_{start\text{-}date}(pma_1)) = \begin{cases} True, \text{if and only if start - date} \\ \qquad \leq \text{current\_date(real - time calendar)} \\ False, \text{otherwise} \end{cases}$$

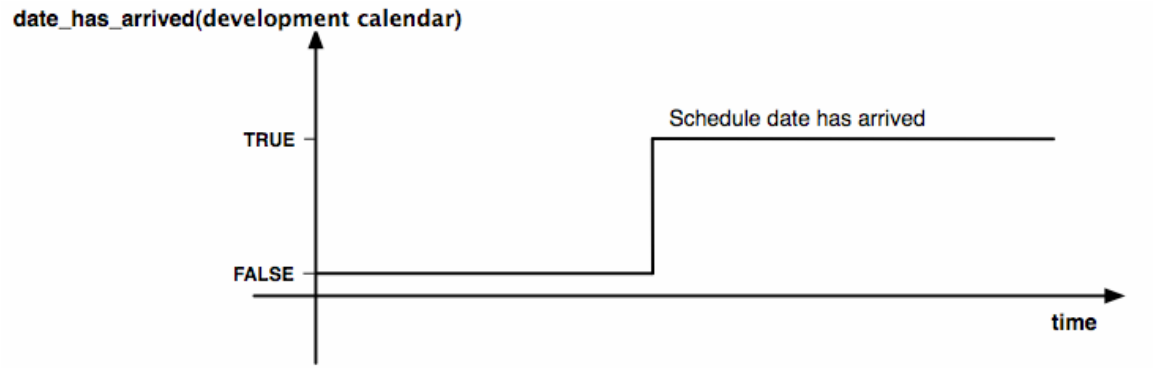


Figure 11: Boolean state value for a calendar event

This kind of Boolean state variable allows involvement of scheduling events in project life cycle management.

We have expanded Boolean state variables from unary to n-ary functions. Now we expand them one more step, to allow sets of tuples as parameters.

Consider $WA_1 \subset WA$, a set of work activities:

$$WA_1 = \{wa_1, wa_2, wa_3\}$$

We define a set-oriented Boolean state variable,

$$any\_is\_being\_performed(WA_1) = any\_is\_being\_performed(\{wa_1, wa_2, wa_3\})$$

$$= \underset{i=1,2,3}{OR} (is\_being\_performed(wa_i))$$

Similarly,

$$all\_are\_being\_performed(WA_1) = all\_are\_being\_performed(\{wa_1, wa_2, wa_3\})$$

$$= \underset{i=1,2,3}{AND} \ (is\_being\_performed(wa_i))$$

Further, projection and selection can be used to specify sets of work activities dynamically as parameters of set-oriented Boolean state variables. For example, John Doe doing usability evaluation in *any* work activity *wa* could be shown using:

$$any\_is\_being\_performed(wa|\prod_{wa\text{-}type,\ developer}(wa) = (usability\ evaluation,\ John\ Doe))$$

Similarly, in certain situations, it might be useful to find all usability evaluation work activities that John Doe is *currently* working on. If the result of such query is a set WA_3, this can be determined using the expression:

$$WA_3 = \{(wa|\prod_{wa\text{-}type,\ developer}(wa) = (usability\ evaluation,\ John\ \text{Doe})) \ \&\ is\_being\_performed(wa)\}$$

3.4.2 Time-based events

To support coordination of events within parallel life cycles it is often not enough to know that a given work activity is currently being performed. It is often more useful to detect when an activity begins and ends. In such a case we are more interested in state changes, such as wa_1 began, than simply state information, such as wa_1 is being performed. To derive this kind of time-based event information from a Boolean state variable, we need a derivative function that can *detect* the occurrence and polarity of each state change in a Boolean state variable with respect to time. We define $\frac{d}{dt}(\text{Boolean state variable})$ as a time-based event which is a signal denoting the *detection* of the corresponding state change. The output of this derivative function, as shown in Figure 12, is discrete signals occurring over time indicating the detection of state changes of the corresponding Boolean state variable.

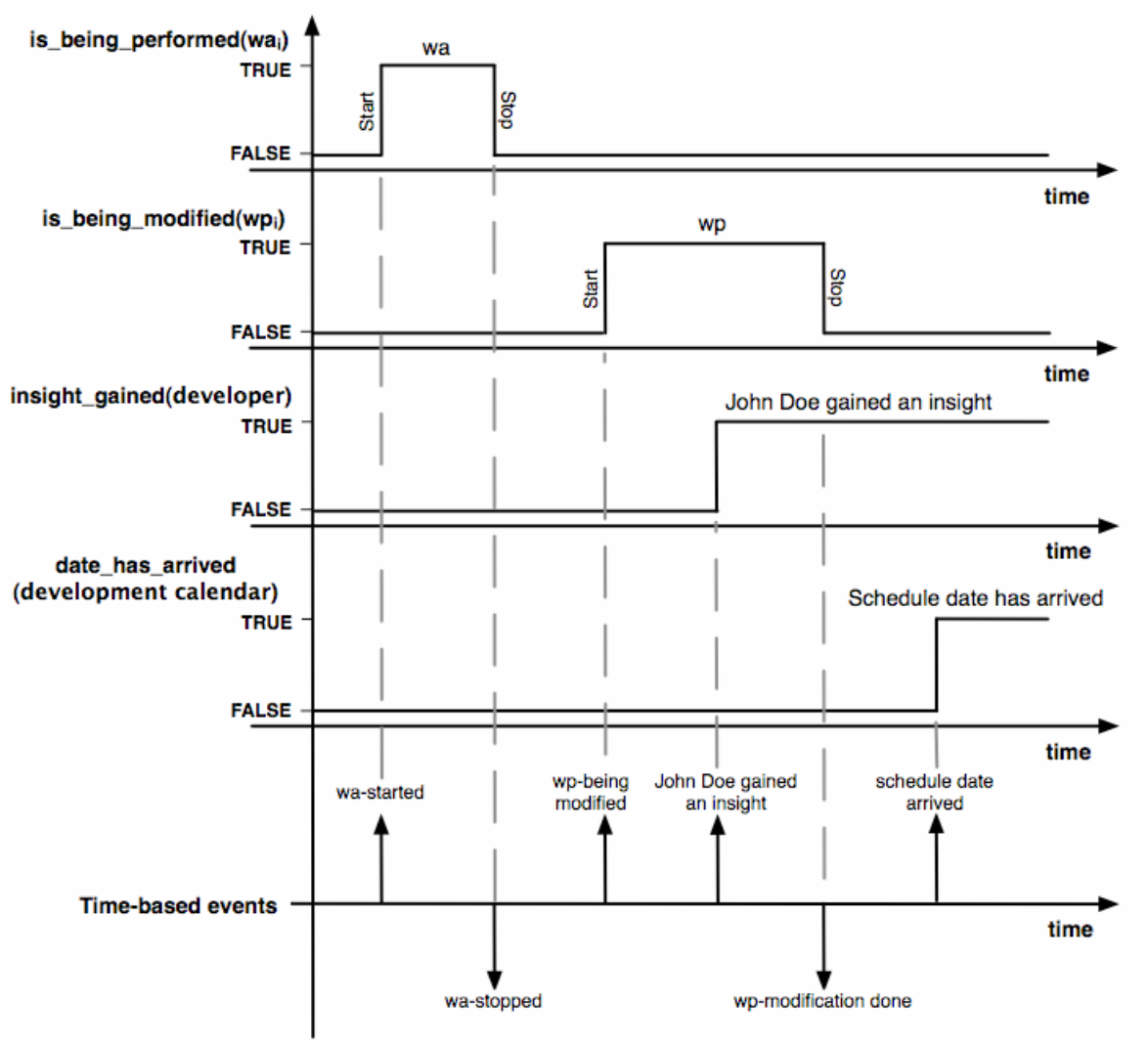


Figure 12: Derivative function of Boolean state variables

Each such time-based event can be named for the state change event it signals. Further, at exactly the time a rising state is detected, the derivative (signal) is positive. Similarly, at exactly the time a falling state is detected, the derivative (signal) is negative. When no state change is detected, the derivative (signal) is zero.

Or, the occurrence of $wa_1\_started$ is in fact the detection of an occurrence of $\frac{d}{dt}(is\_being\_performed(wa_1)) > 0$. Similarly, the end of a work period on wa_1 (shown in Figure 13), can be represented by the occurrence of the time-based event, $wa_1\_stopped$ which is the detection of an occurrence of $\frac{d}{dt}(is\_being\_performed(wa_1)) < 0$.

Since these derivatives have non-zero values as "spikes" or singular points on the time axis, they are event *signals*, rather than states. Thus, as shown in Figure 13, we can use the occurrence of the time-based event, *wa1_started*, to signal the beginning of a work period on *wa1*. We use

$$a \xleftarrow{S} b$$

to show that "a is signaled by the occurrence of b" as shown here:

$$wa_1\_started \xleftarrow{S} (\frac{d}{dt}(is\_being\_performed(wa_1)) > 0)$$

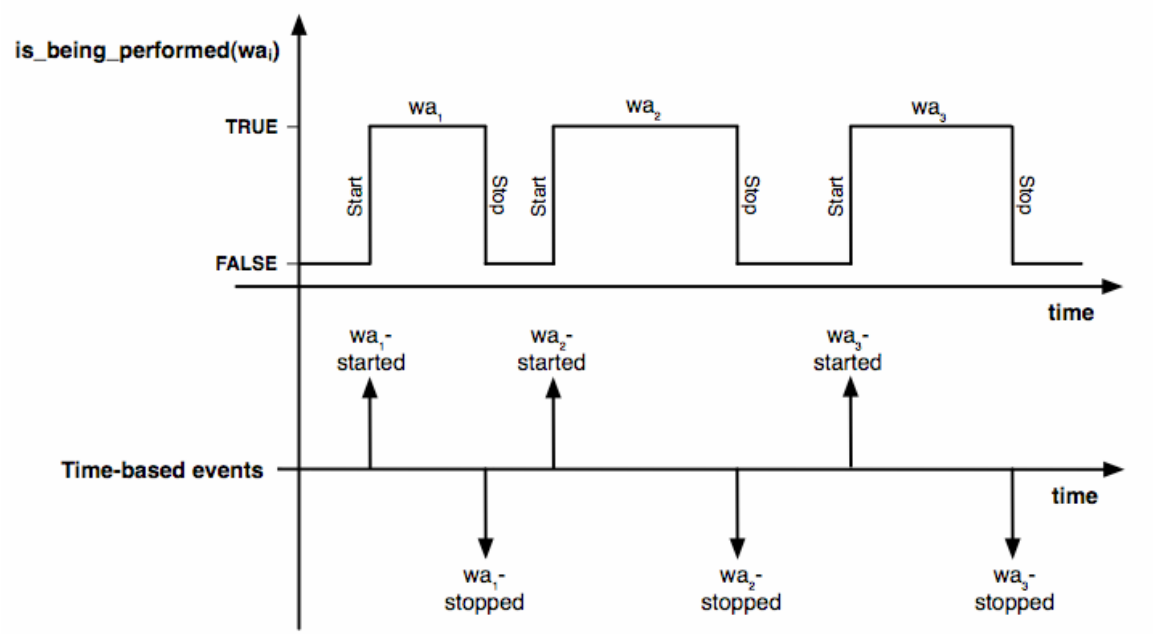


Figure 13: Time-based events for work activities

As an example of time-based events for set-oriented Boolean state variables defined dynamically with projections and selections, we can show:

$$John\_Doe\_started\_usability\_evaluation \xleftarrow{S} \frac{d}{dt}(any\_is\_being\_performed(wa|\prod_{wa\text{-}type,\ developer}(wa) = (usability\ evaluation,\ John\ Doe)) > 0$$

This time-based event function detects the point in time when the *is_being_peformed* Boolean state variable rises to a true value for any *wa* that has John Doe as a developer and usability evaluation as the work activity type.

3.4.3 Trigger events

A trigger event is a time-based event that is useful for signaling the need to communicate, in order to coordinate, or synchronize work activities to support change management or constraint and dependency enforcement among work activities in the two life cycles, or to notify developers working on other *related* work activities such as functional testing in the SE life cycle to, say, attend the usability evaluation session to watch for any functional problems (bugs). Each trigger event signifies a need to invoke a particular type of dependency among work activities. As shown in Figure 14, if two work activities *wa-source* and *wa-target* are related to each other via a relationship *r*, then the trigger event *te* signifies the point in time when *r* should be enforced. The motivation for such relationships and the need for enforcement are discussed in the next section.

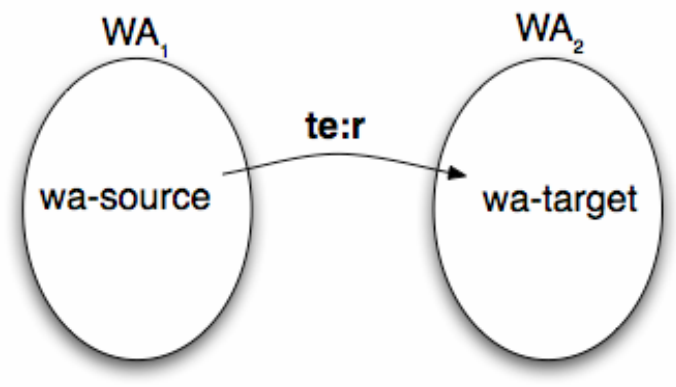


Figure 14: Dependency relationships among work activities

3.4.4 Motivation for mapping components

3.4.4.1 Dependency relationships

Many work activities in each life cycle have dependency relationships, *R*, with other work activities in either life cycle. We identify any two work activities, be it analysis, design, development, or evaluation and regardless of which life cycle either appears in, to be related or dependent on one another if work (or other events) in one work activity implies the need to consider corresponding work (or other events) in the other work activity. For example, if the usability engineering role is undertaking a task analysis work activity, a new user task implies the need to at least consider additional functionality to support it on the system side; and, vice

versa, changes in functional analysis imply the need to consider corresponding changes in task analysis.

Since these dependency relationships are often independent of which developer is performing either work activity, we think of them as being more dependency relationships between work activities than between specific developers. Therefore, technically, in cases where communication is independent of the developer, the communication is between projections of work activities. Thus, we can view the dependency relationships between work activities in the two life cycles as shown in Figure 15.

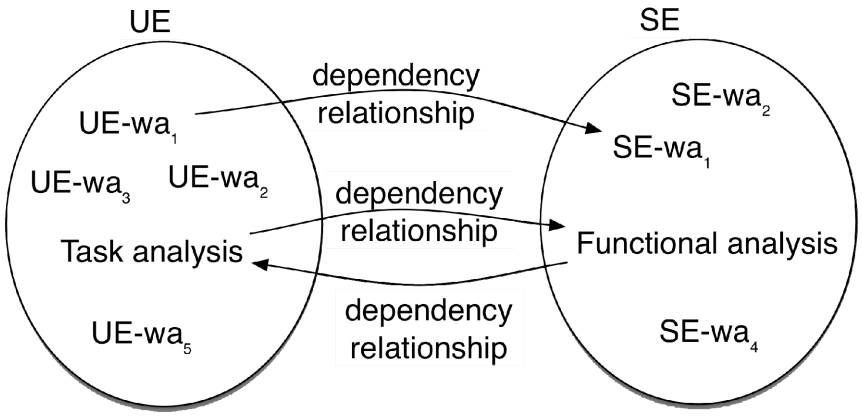


Figure 15: Dependency relationships between work activities and/or their projections

Apart from dependency relationships between work activities in the two life cycles there is also a need for dependency relationships *within* a development life cycle. For example, in the UE life cycle, work on a task analysis work activity may be related to scenario development work activity as most key tasks need to be enacted using scenarios. Similarly, the scenario development work activity is in turn related to work activity for developing benchmark tasks as most key scenarios need to be translated to benchmark tasks for the usability evaluation work activity. Hence, the relationships look like those shown in Figure 16.

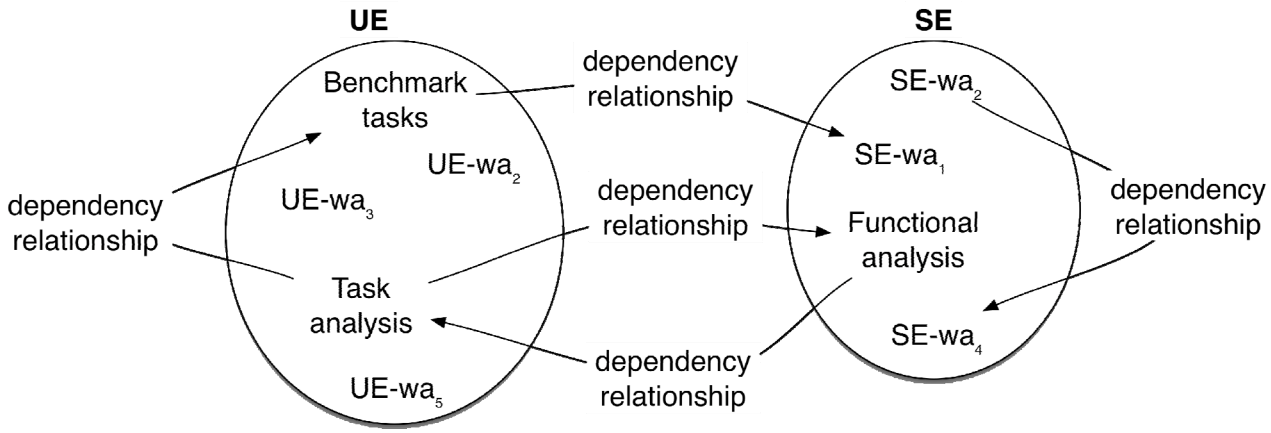


Figure 16: Dependency relationships between and within the two life cycles

3.4.4.2 Communication needs

Identifying the need to consider changes, for example, is not useful unless that need can be communicated to where it should be considered within the two life cycles. Thus, when a UE developer performs work on the task analysis work activity, there is a need to *communicate* this fact to the SE developer who later works on the functional analysis work activity. Thus, dependency relationships often indicate communication needs as shown in Figure 17.

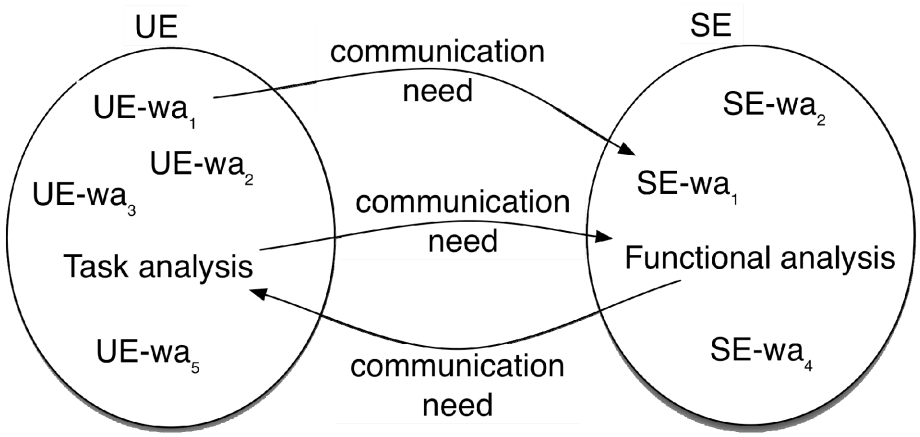


Figure 17: Communication needs between work activities and/or their projections

In simple terms, communication adds message content and is the way that dependency relationships are enforced within Ripple. We often need this communication to be signaled by the start and end points of work activities which have dependency relationships with other work activities. These start and end points are trigger events, which are in turn specific time-based events (Section 3.4.2). Each trigger event, depending on the type of work activity, signals the

need for enforcing a particular type of dependency relationship. For example, the start of a task analysis work activity on the UE side could signal a "heads-up, potential changes coming" message, whereas the end of a task analysis work activity could signal a "please synchronize functional analysis with new task analysis" message.

3.4.5 Structure of mappings

To establish an abstraction for a method to satisfy the needs for communication described in the previous section, we describe a mapping *M* as shown below. The mappings of this formal description model will become messages in the implementation framework and its instances.

$$M = WA_{source} \times TE \times WA_{target} \times R$$

i.e. $m = \langle wa_{source}, te, wa_{target}, r \rangle$

where WA_{source} is a set of work activities where trigger events signify the need to communicate, *TE* is a set of trigger events, WA_{target} is the set of work activities that require consideration for additional work as a result of work on work activities from WA_{source}, and *R* is a set of dependency relationships among work activities. For the example about task analysis and functional analysis, the mapping would look like:

m_1 = <*desktop-client-task-analysis, Jon_Doe_completed_hierarchical_task_analysis, functional-analysis, consider_changes_to_functional_specifications* >, where

Jon_Doe_completed_hierarchical_task_analysis= $-\frac{d}{dt}(is\_being\_performed(wa_p))$, and

wa_p = <*desktop-client-task-analysis, UE, task-analysis, hierarchical task analysis, task inventory, usability analyst, John Doe*>

In an implementation of Ripple, when Jon Doe completes the hierarchical task analysis of the desktop client, we envision him being able to change the status of wa_p to be completed. And this would automatically send and queue a message to the effect "UE role Jon Doe completed hierarchical task analysis on desktop client, consider revising functional specifications to ensure consistency" to be viewed by the next person who works on functional specifications within the SE life cycle.

3.5 Summary

In this chapter, we described an abstract formal notation to chart the development space of interactive-software development. Using the concepts provided here it is possible to construct a framework that describes the interaction between the SE and UE life cycles, associated roles, and the environment and tool support required for supporting an interactive-software development endeavor. We describe one such framework, called the Ripple Implementation Framework in the next chapter.

4 Chapter Four: Ripple Implementation Framework

4.1 Introduction

The Ripple Implementation Framework (RIF) reifies the Ripple Description Model to describe specifically the environment, tool support, entities and various components involved in the development of interactive systems. The RIF, as shown in Figure 18, is expressed at a level of detail that is useful for developers to adopt and employ manually for a particular project context or as a framework on which to design an automated software system (a Ripple implementation) to manage the communication required between the two life cycles. In this chapter we describe how such an automated software system might work.

As discussed in Section 1.8 the RIF is agnostic with respect to any particular SE or UE development life cycle such as waterfall model (Royce, 1970) or the Star Life Cycle (Hartson and Hix, 1989). This framework provides mechanisms for developer roles within each life cycle to communicate with one another to allow for coordination, collaboration, synchronization, and change management while functioning independently. Also, the RIF embraces:

- facilities for developer definition of the abstract and component work activities, and resulting work products from each life cycle in the integrated development effort;
- facilities for developer's definition of mappings to capture dependencies, constraints, and relationships among different entities in the development space for interactive software;
- idea of a constraint subsystem that includes mappings, triggers, and relationship enforcement via messages, to respect the constraints and dependencies between and within the two development life cycles; and
- the abstraction to facilitate an instantiation to suit almost any project and resource context.

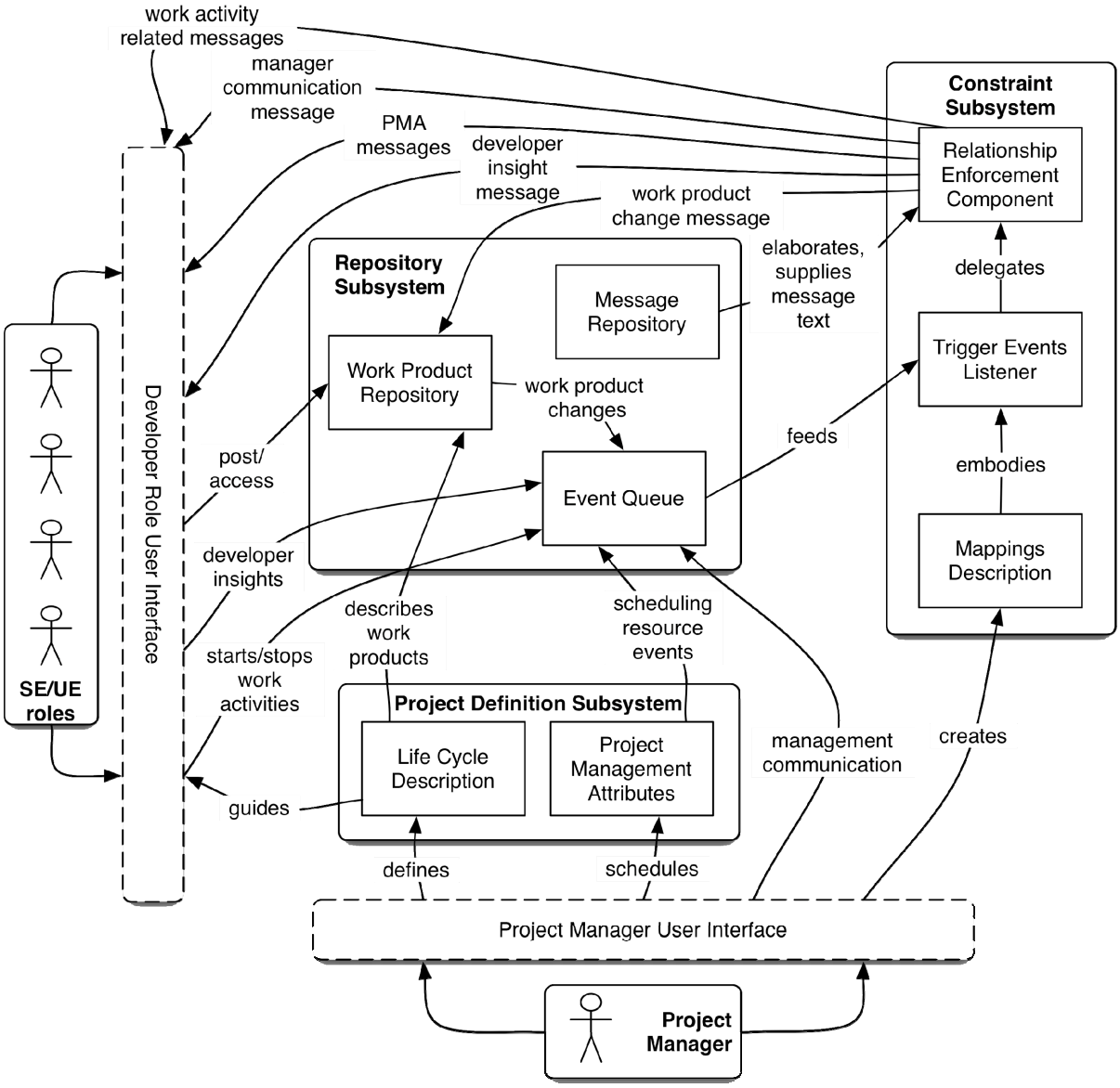


Figure 18: The Ripple Implementation Framework

4.2 Components of the Ripple Implementation Framework

In the following sections we provide a high-level description of the RIF.

4.2.1 Workings of a software-based Ripple implementation

We envision two user interfaces within a Ripple implementation: one for the developers and one for the project managers. For a given project, the project manager uses the manager interface to declare and specify all the required management attributes. These declarations act as specifications for the behavior and structure of the various entities of a project, including the

work activities and stages or phases of the project. Examples and discussion of these declarations is provided in Section 4.2.2. Similarly the developer roles use another user interface to access the development environment which facilitates communication and provides tool support for interactive-software development. For example, a UE developer John Doe can login to the system, "start" working on a task analysis by accessing a UI widget that supports such an endeavor, "create" a hierarchical task list document which will be stored in a work product repository, and "post" an insight to all developers about the need for the system being developed to support a new task that he discovered as part of this analysis. The Ripple implementation will automatically detect the fact that John Doe started task analysis, the work product repository will automatically detect the creation of this new work product, and Ripple implementation will allow the developer to create an event for his or her insight, respectively. Upon creation, these three events will be sent to the event queue component and acted upon appropriately. For example, if a dependency relationship exists between the UE's task analysis and SE's functional analysis: "every task in UE role's HTA must have one or more corresponding functions to support the task on the backend", the system automatically sends a message to the functional analysis work activity in SE. This message will be waiting for when the SE role logs in through the developer interface and starts that activity. Similarly, when John Doe sends the insight about the need for a new task, the system automatically sends messages to all developers who work on task analysis and this message will be delivered immediately. We describe the modules and the workings of such an envisioned project environment in the following sections.

4.2.2 Project declarations via the Ripple project definition subsystem

Using a project manager interface (as shown in Figure 18) a project manager accesses the Ripple project definition subsystem to specify the component parts of a project. This module contains two parts: life cycle description component and project management attribute component. Within the life cycle description component, the project manager declares the SE and UE life cycle types, work activities to be conducted as part of the two life cycles, developer roles and names for these work activities, work product names and formats resulting from these work activities. Within the project management attribute component, the project manager declares constraints (e.g. schedules) and resources allocated for the project. We envision a library of customizable

templates to aid the project manager in declaring each of these entities. We describe these two parts in detail here.

4.2.2.1 Life cycle description

The life cycle description component is an entity using which a project manager can declare work activities (individual and block) and their attributes for both the SE and UE life cycles. The entries in this module guide the SE and UE roles in the form of a roadmap of activities and associated attributes that need to be undertaken during the course of a project. We envision this life cycle description to be expressed in a structured manner using a data definition language such as XML, which would then be interpreted by the Ripple software implementation. Using a project managers' UI to this XML description, one can declare the name of a work activity (e.g. initial requirements analysis), the life cycle it belongs to (e.g. SE), the type of work activity (e.g. analysis), the technique to be used (e.g. user interviews), work product expected (e.g. high-level software requirements), developer role who should undertake this activity (e.g. requirements engineer), and the name of the developer who is assigned this job (e.g. John Smith). As mentioned above, for a common type of work activity such as the one described in this example, one would have a pre-defined template (that can be customized) for use.

4.2.2.2 Project management attributes

The PMA component is an entity using which a project manager can declare the available schedules and resources for the project. We envision a calendar-style UI for a project manager to specify the various due dates for each work activity or block of work activities and the total resources that are allocated to each of those activities. For example, for a requirements analysis activity, the project manager may assign a due date (e.g. 8/24/07), two developers to help John Smith (e.g. Jane Doe and Jack Brown), and a project room to conduct user interviews (e.g. Room 123 from 8/12/07 to 8/24/07). We envision this component to have mechanisms to automatically detect calendar-based events and sending them to the event queue (Section 4.2.4.2) to be acted upon by the trigger event listener (Section 4.2.3.3).

4.2.3 Ripple constraint subsystem

The constraint subsystem is a module that would include functionality to represent, observe, and enforce the various dependency relationships among different entities between the two life cycles

during development. In the following sections we provide background, and envisioned workings of a software-based Ripple implementation for this subsystem.

4.2.3.1 Background

A constraint is a "relation that must be maintained" (Borning and Duisberg, 1986). Such relations are generally enforced by "delegating to the constraints solver the task to satisfy them automatically" (Kwaiter, Gaildrat and Caubet, 1998). In other words, a constraint-based system is one that automatically updates a predefined set of relations and dependencies between different entities when a change occurs in one or more of such entities. Constraint-based systems were traditionally used to specify declaratively the relative layout of interface objects according to pre-specified rules (Szekely and Myers, 1988). Some of the other important applications for constraint-based systems include:

- specification of relations (constraints) among the user interface objects that should be maintained upon resizing a given UI window (Mugridge, Hosking and Grundy, 1996), (Chok and Marriott, 1995),
- visual representation of simulation algorithms (Ege, 1988),
- automatic updating of (to make consistent) multiple views representing the same data when the objects in one of the views is changed (Borning and Duisberg, 1986), and
- triggering of events based on changes made to objects in a dataset (Bharat and Hudson, 1995).

This last application of constraint-based systems is the one that corresponds to our focus. Different time-based events (discussed in Section 3.4.2) in the development space have the potential to trigger other events that need to be performed to regain stability in the design that was lost due to the occurrence of the said time-based event. In order to address this need, the constraint subsystem has three components for: mappings description, observing trigger events, and enforcing relationships.

4.2.3.2 Mappings description

The mappings description component is an entity using which the project manager can declare the different relationships that exist among various entities within the development space. For example, consider the relationship between the SE role's functional decomposition work product

and the UE role's hierarchical task list work product: a change to one of these work products requires at least a consideration of change to the other. Therefore a project manager, using his/her UI, can declare a mapping between these two work activities to include the source work activity (e.g. HTA by UE role), trigger event that perturbs the design space (e.g. new task description added to HTA by UE role), related work activity elsewhere in the design space (e.g. functional analysis by SE role), and the type of relationship (e.g. every task in UE role's HTA must have one or more corresponding functions to support the task on the backend).

4.2.3.3 Trigger events listener

We envision the trigger event listener to be an automated software agent that would constantly monitor the event queue (Section 4.2.4.2) for different time-based events that signify the invocation of the need to enforce a relationship. These events include occurrences such as a developer gaining an insight about the software system being developed, start or stopping of a work activity, creation or modification of a work product, reaching of a calendar date, etc. For each time-based event arriving at the event queue, the trigger events listener checks the mappings description to identify the corresponding relationship, and delegates the enforcement of that relationship to the relationship enforcement component (Section 4.2.3.4) by passing to it the event and its corresponding relationship. For example, in the case of the UE role creating a new task in the HTA, the module upon verifying the existence of a relationship wherein the SE role is required to update their functional decomposition work product, informs the relationship enforcement component to notify the SE role about this change.

4.2.3.4 Relationship enforcement component

We envision the relationship enforcement component to be a software-based communication agent in the RIF. Upon notification from the trigger event listener (for example, via a software call), the relationship enforcement component extracts the relationship that was passed by the trigger event listener and retrieves the corresponding message description (text) from the message repository (Section 4.2.4.3) and sends it to the appropriate entity of the target work activity or developer role as described in the relationship. For example, when an event arrives at the event queue marking the case of the UE role's adding a new task to the HTA, the relationship enforcement component extracts the relationship "every task in UE role's HTA must have one or more corresponding functions to support the task on the backend" and identifies the

corresponding message in the message queue, which includes a more human understandable description and context. This message is then communicated to the functional decomposition work activity in the SE life cycle. Therefore the next time a SE role attempts to do the functional decomposition work activity this message is shown.

4.2.4 Ripple repository subsystem

The RIF has three storage components, each of which maintains a repository of different development artifacts. Each of these components is described here.

4.2.4.1 Work product repository

The RIF represents the various work products of the combined design process in a single repository with each of the SE and UE roles having two separate views to this dataset. We envision this to be a shared design representation where each developer roles can access appropriate work products created by colleagues in their role or the counterpart role via a developer interface to ensure design consistency across the overall process. Developers are required to post new work products created at the end of each work activity here. We envision the work product repository to be a document versioning system where access is controlled and any change made to the documents are logged and incremental versions maintained. The Ripple implementation of this repository would also have mechanisms to detect and queue events as and when time based events for work products such as *work product created* or *is being modified* occur. Once detected, these events are sent to the event queue to be acted upon by the trigger event listener.

4.2.4.2 Event queue

We envision the RIF to have a component that would maintain a list of events that are generated by the various entities during a development project. This component gathers inputs, in the form of events, from developer roles, the work product repository, PMA component, project managers, and the Ripple implementation system itself. These events are in turn acted upon by the trigger event listener. An example of a developer role generated event could be deliberate messages sent by a particular developer to others in the project to, say, share an insight about the project. An example of a Ripple implementation system generated event could be the system detecting that a particular developer role has started a work activity (discovered by the fact that

the developer role just initiated that work activity via the developer UI) and sending that event to the event queue.

4.2.4.3 Message repository

The message repository is an entity using which the project manager can define different messages to go with the different relationships described in the mapping description component. We envision these messages to be declared at the start of the project from preexisting templates which can be customized to suit a particular project. We discuss a few message types that could be common in a project.

"Work activity performed" message

The work-activity-performed message informs the two roles about the completion of a particular activity or phase in the life cycle and contains a link to the relevant products (in the work product repository) of this development stage. Developers in the other life cycle or developers at a different stage of the project within the same life cycle can use this link to view the product artifacts. This message is generally used when the type of communication is purely informational and no corresponding action is necessarily required. For example, when the usability engineers complete the initial screen layouts or the derivation of the conceptual metaphor for the interaction design, they can send this type of message to the software engineers to let them know about the progress and to allow them to peruse the results. Another example for this type message is when a usability engineer wants to give a "heads-up" to the software engineers about completion of a screen design so that the SE role can start making plans to implement it, pending evaluation.

"Synchronize activity" message

The synchronize-activity message informs about the need for a joint activity, or at least the need to get together to collaborate directly on a problem, by both the SE and UE roles. In other words, this message addresses the synchronization need for activities that require a combined presence of the two developer roles. For example, when the usability engineers plan an evaluation session, they can send this type of message to the software engineers to request them to be present (to help argue the case for required changes in the user interface when the SE role sees the users having problems). Similarly, early systems analysis and ethnographic study activities that require

joint presence can be arranged using this kind of message (to help identify the broader constraints of the project and get the overall context).

"Work product modified" message

The work-product-modified message is used to enforce the consistency of data objects in the work product repository. This message informs the developer of the need to perform a consistency check on products of the two development roles. For example, in the object oriented development paradigm, this type of message can be initiated after the use-case specifications phase in the SE life cycle or the usage scenario descriptions in the UE life cycle. Since these two stages of development concentrate on two aspects of the same issue: interaction between the system and user, there is a need to ensure that they are consistent. Another important example for the need for consistency is after the usability specifications phase in the UE life cycle and functional requirements in the SE life cycle. A consistency check message is required here to initiate an analysis that ensures that these specifications are supportable by the functional core (and to discuss alternatives if not supportable or negotiate for middle ground). This type of a message is used to enforce such necessary consistency checks.

"Insight gained" message

The insight-gained message is used by a developer role to inform other developers of insights gained in one part of the design and the potential effects of that insight in that and other parts of the design. This is perhaps the most useful message in the development of interactive systems because of the potential for constant and frequent changes in the products during the development life cycles. As an example, this message can be used when a new task is identified by the UE role, and that new addition should be communicated to other development activities within the UE role and to the SE role. Upon the receipt of the message by the SE role, efforts can be made to incorporate the necessary functions in the functional specifications to support the corresponding task. These updates in the functional specifications, in turn, can cause more insights and therefore trigger further changes in various dependent stages' products in the integrated life cycle.

"Calendar date arrived" message

A calendar-date-arrived message is used to announce upcoming schedule dates for the project. Examples of such a message include informing the development roles of an upcoming project deadline or the need for completing a particular work activity for review.

4.3 Potential Downsides of the RIF

The RIF has potential for the following downsides due to the various overheads and additional tasks that arise because of the coordination of the two life cycles:

- Increase in time or cost of the overall software development life cycle;
- Additional effort required by the roles in each life cycle for maintaining the design representation framework;
- Additional effort required for coordination of various activities and schedules;
- Need for stricter verification process than conventional processes to enforce the various synchronization checkpoints during the development effort; and
- Resource overhead to carry out all the above mentioned drawbacks.

However, we believe that not all of the above listed factors will materialize in all cases and situations where the RIF is instantiated. Also, some of these potential downsides such as the need for stricter verification could actually turn out to be advantages because the greater the verification effort the higher the probability of discovering potential problems early on in the life cycle. Even with the incidence of one or more of these downsides in a project, we believe that the advantages of using a framework like RIF to connect the SE and UE life cycles will outweigh any potential downsides.

4.4 Summary

In this chapter we described the Ripple Implementation Framework, which describes the key components, potential tool support, and development environment necessary to support an interaction-software development effort. One of the key components of this framework is a storage subsystem, which acts as a shared design representation where UE and SE roles can store, access, share, and manipulate the various work products that are created during a development effort. Another important component of this framework is the constraint subsystem

that can be instantiated to record, propagate, and enforce the various dependency relationships that exist among the various entities between the SE and UE life cycles.

5 Chapter Five: Ripple Implementation Instance

5.1 Introduction

As described in Chapter Four, the Ripple Implementation Framework describes the environment, tool support, entities and various components involved in the development of interactive software systems. As mentioned in that chapter, one can manually instantiate the RIF, if necessary to suit the constraints and context of a project. In this chapter we describe one such manual instantiation for an academic setting, as a proof of concept and background for an exploratory study. We used this instantiation to facilitate the team project portion of a cross-pollinated SE-UE course offering.

We requested and got approval from the Department of Computer Science at Virginia Tech for a coordinated offering of Dr. H. Rex Hartson's graduate-level usability engineering (CS/ISE 5714) and Dr. James D. Arthur's graduate-level software engineering (CS 5704) courses. These courses were offered in the fall of 2006 as a testbed for exploratory study to evaluate Ripple and to expose students to the concepts of connecting SE and UE life cycles. In this joint offering students in the SE class were trained for the SE role and students from the UE class were trained for the UE role. Students who were enrolled in both classes were trained as dual experts. As part of an exploratory study, all teams in this joint offering were assigned to different development conditions (discussed in Section 6.3.1) and were required to work on a semester-long project to develop a software system to facilitate the annual plant sale for the Horticulture Club of Virginia Tech. Two of the eight teams from this study used this Ripple Implementation Instance as their development condition. More details of this exploratory study are presented in Chapter 6.

5.2 Components of the Ripple System Implementation Instance

As described in Chapter Four, the Ripple Implementation Framework has the following high-level modules: project definition subsystem, repository subsystem, and constraint subsystem. We describe the instantiation of each of these entities and their components for a classroom setting in the following sections.

5.2.1 Project definition subsystem

The project definition subsystem in the RIF is used to declare the various project management entities such as life cycles to be employed, work activities to be used, PMAs available etc. For this instantiation the experimenter (author of this dissertation) played the role of a project definition subsystem and manually declared these attributes on paper. Specific details of the various components of this instance are as follows.

5.2.1.1 Life cycle description

For this Ripple Implementation Instance the project specifications section of the syllabi of the two classes, where the processes the teams need to adopt were described, acted as the life cycle description. The SE and UE roles checked these documents for instructions on what upcoming work activities are and what resulting work products were expected of them as deliverables. Figure 19 and Figure 20 show the high-level declaration of the life cycle description and project schedule for the two processes (each of these activities listed here had detailed specifications accompanying them that are not shown here). Appendix G contains a more detailed project schedule.

SE life cycle description and resulting work products	Planned Completion Date
▼ **Product Overview** • Provide overview of what product does • Describe the problems the product solves • Give high-level description of major functionality • List stakeholders	**5th Sep 2007**
▼ **Software Requirements Specification** • Introduce system and discuss scope • Identify intended audience of the system • Describe overall purpose and functionality of system • Describe the behavioral aspects of system • Describe functions, non-functions, interfaces • List use cases, prototype/models, traceability matrix	**19th Sep 2007**
▼ **High-level Design** • Describe architecture of system and supporting diagrams • Justify choice of architecture with appropriate diagrams	**21st Sep 2007**
▼ **Low-level Design** • List major components of system • Show relations among these components using diagrams	**21st Sep 2007**
▼ **Implementation** • Implement to code • Integrate user interface components	**7th Nov 2007**
▼ **Acceptance testing** • Demo to client representative • Make changes based on client feedback and available resources	**16th Nov 2007**

Figure 19: SE life cycle description

5.2.1.2 Project management attributes

For this Ripple Implementation Instance the deliverable deadlines for the semester-long projects the teams conducted were used as project management attribute declarations. Because of the academic setting, there was no explicit assignment of any resources to the teams. The teams acquired their own resources for the project and used the meeting rooms in the university library and other conference rooms. The project management attributes for the two life cycles are shown in Figure 19 and Figure 20:

SE life cycle description and resulting work products	Planned Completion Date
▼ **Product Concept Statement** • Provide name of system and client information • Describe users and how the system will help them • Provide technical summary of target application system	**1st Sep 2007**
▼ **Systems Analysis** • Prepare and execute ethnographic field studies • Describe target application • Create product concept statement • Construct business process model • Perform needs analysis • Conduct user analysis • Perform task analysis • List usability goals • Specify constraints	**14th Sep 2007**
▼ **Design** • Create usage scenarios • Design conceptual metaphors • Sketch initial screen layouts • Gather client feedback • Construct prototype • Revise screen designs • Write custom interaction design style guide	**28th Sep 2007**
▼ **Usability Specifications** • Create benchmark tasks and descriptions • Set usability targets • Tabulate usability specifications	**17th Oct 2007**
▼ **Formative Evaluation** • Create evaluation measures • Record critical incidents • Diagnose usability problems • Perform cost-importance analysis	**2nd Nov 2007**
▼ **Redesign** • Revisit necessary activities • Assemble user interface and backend	**16th Nov 2007**

Figure 20: UE life cycle description

5.2.2 Ripple constraint subsystem

For this Ripple Implementation Instance the experimenter played the role of the Ripple constraint subsystem. The experimenter kept track of calendar dates, changes to work product repository, start and stop of work activities by various student developers, and enforced the various dependency relationships via email (instead of automated) messages. Specific details are as follows.

5.2.2.1 Mappings description

For this Ripple Implementation Instance mappings signifying coordination between SE and UE roles were listed based on anticipated client meetings with each individual role. As described in Chapter 6, the meetings of each developer role with the client were carefully controlled and scripted. As and when a particular role scheduled a meeting slot with the client, a coordination style mapping was recorded in the mappings description maintained by the experimenter. A chronologically ordered list of coordination mappings is shown in Table 1. All relationships in this instantiation were mapped to developer roles and not to work product repository or work activities for ease of execution.

Source	Trigger	Relationship
UE initial meeting with client	UE schedules appointment with client	SE presence or representation is encouraged at this meeting
SE initial meeting with client	SE schedules appointment with client	UE presence or representation is encouraged at this meeting
SE requirements validation meeting with clients	SE schedules appointment with client	UE presence or representation is encouraged at this meeting
UE low-fidelity prototype walkthroughs with clients and club members	UE schedules appointment with client	SE presence or representation is encouraged at this meeting
UE formative evaluation with clients and club members	UE schedules appointment with client	SE presence or representation is encouraged at this meeting
SE customer acceptance meeting with client representative	SE schedules appointment with client representative	UE presence or representation is encouraged at this meeting

Table 1: Coordination mappings used in this Ripple Instance

Constraint and dependency mappings were created dynamically as and when a new work product was posted by the SE or UE roles. The reason for this on-the-fly after-the-fact mappings creation was due to the fact that the SE class did not mandate a particular approach to developing the functional core (e.g. structured programming, Object-Oriented development, etc.) and therefore the actual software engineering work activities being undertaken were not known until after they were executed and the resulting work products were posted as project deliverables. When a work product was posted the work product was analyzed to find dependences between that and the counterpart UE role's work products and any mappings discovered were recorded. These two-way mappings are shown in Table 2 in no particular order and were all triggered as and when a deliverable was posted.

Work product	Related to:
SE use cases/requirements	UE usage scenarios
UE screen prototypes	SE data dictionary attributes
SE software requirements/functions	UE hierarchical task analysis and task objects (as identified by scenario annotations)
SE data flow diagrams	UE task sequences
UE business process modeling	SE software interface description (external machine interface)
SE software interface description for human interface	UE usability goals
SE state transition diagrams	UE state transition diagrams
SE software architecture	(supports) UI features such as undo, cancel, auto-field-completion, drag and drop, etc.
SE low-level design	UE hierarchical task inventory
UE cost-importance table	SE resources available

Table 2: Dependency mappings between the two life cycles

Mappings among the work activities in the same life cycle were not listed or defined for this instantiation.

5.2.2.2 Trigger events listener

We used a Wizard-of-Oz approach to instantiate the trigger event listener. The experimenter played the role of this component behind the scenes by constantly observing the progress of each team and "triggering" necessary events to keep the two life cycles connected. For example, whenever a mapping was observed, necessary action trigger event was "fired". An example is provided in the following subsection.

5.2.2.3 Relationship enforcement component

Similar to the trigger event component, we used a Wizard-of-Oz approach to instantiate this component. The experimenter "enforced" each relationship by "firing" an email message to the relevant team. These email messages contained information about what the receiving developer role should do to ensure the two life cycles are in agreement. For example, for the coordination type mapping such as UE role conducting client walkthroughs of their low-fidelity prototypes requiring SE representatives being present, an email was sent to the SE role every time the UE role scheduled a walkthrough meeting with the clients asking them to have representatives at this meeting. Examples of actual email messages that were sent during the duration of the project are shown in Table 3.

Team UE-**, Your counterpart team, SE-**, is scheduled to meet the project clients on Friday, Sep 15th at 12:00 noon in McB618, to conduct requirements validation. It is recommended that you have representatives from your team at the meeting to ensure that you are on the same page with the SE team. Also, this will be a good opportunity to see your SE counterpart team's requirements analysis and gauge the direction they are taking. Good luck, -Project directive module
Team **, The UE experts in your team are scheduled to meet the project clients on Monday (Oct 30th) from 3.00PM to 4.00PM in a place TBD, to conduct their formative evaluation. It is recommended that you have representatives from the functional side of your team at this meeting. This may be a good opportunity to see your team's design in action as users perform key tasks using a hi-fi prototype of your system. Good luck, -Project directive module P.S.: You will be informed of the location of these meetings as soon as it is decided.
Team **, The UE experts in your team are scheduled to meet the project clients on Monday (Oct 30th) from 3.00PM to 4.00PM in a place TBD, to conduct their formative evaluation. It is recommended that you have representatives from the functional side of your team at this meeting. This may be a good opportunity to see your team's design in action as users perform key tasks using a hi-fi prototype of your system. Good luck, -Project directive module P.S.: You will be informed of the location of these meetings as soon as it is decided.
Team **, Very soon you will be performing your cost-importance analysis. It is recommended that you take inputs of your SE members on the cost-to-fix attribute for your usability problems as they are the ones who will make these changes on the functional system (and not the prototype). Good luck, Project Directive Module
UE-Team**, Your counterpart SE team is doing a code/preliminary demo with Mr.*** ******* (GTA for SE class). It is recommended you have a representative from your UE team present there. This demo might provide some insights about constraints the functional parts of the system might have on your UI designs. Time: 4:00 to 4:30 pm on 10/25 Location: MCB 133 Good luck, --Project directive module

Table 3: Examples from message repository

5.2.3 Ripple repository subsystem

For this Ripple Implementation Instance, we used a combination of online and paper-based devices to realize the repository subsystem. Specific details are as follows.

5.2.3.1 Work product repository

In this Ripple Implementation Instance, we used a forum-based online group email as work product repository for the teams. Each team was provided with a Google Groups™ email ID. Emailing this ID transmits the message to all members of the group. This group email ID has functionality to maintain threads of email and to access all email posted to the group via a forum-styled web interface (Figure 21).

Figure 21: Group email interface as a work product repository

An email thread titled "Deliverables" was created by the experimenter and all teams were required to post their completed work products as replies to that thread. In other words it was possible for any member of the group to log into the web interface for the group and access

deliverables from this thread. This served as a work product repository for the team members (Figure 21).

5.2.3.2 Event queue

For this Ripple Implementation Instance the experimenter sent emails in real time (as and when a trigger event was observed) and no queue was maintained.

5.2.3.3 Message repository

For this Ripple Implementation Instance the experimenter maintained a set of predefined messages in a text document that were used as email content by the relationship enforcement instance. This list was updated as and when new relationships were discovered. A subset of this message repository is shown in Table 3.

5.3 Summary

As described previously developers can adopt and employ an instantiation of the Ripple Implementation Framework manually for a particular project context or implement an automated software system (a Ripple implementation) to manage the communication required between the two life cycles. In this chapter we described a manual instantiation where we enforced mappings at a much courser granularity. For example, instead of mapping trigger events at the level of a completion of a work activity as illustrated here, it is possible, if desired, to fire events when individual components of a project deliverable are completed or if these deliverables are updated. The only requirement for such lower granularity event detection is a work product repository that can capture such changes to the work products. Similarly unlike in this instantiation, it is possible to map relationships within the same life cycle also if needed.

6 Chapter Six: An Exploratory Study

6.1 Introduction

The next step in this research was to evaluate the Ripple concept as embodied in the Ripple implementation instance. In this chapter we provide a detailed description of an exploratory study we conducted in a classroom setting, discussion on software metrics, how we derived our metrics, team distribution, clients, and other aspects pertaining to this study.

6.1.1 Full summative study not feasible

Theoretically, the most scientifically rigorous way to evaluate a software development framework such as the Ripple Implementation Framework would be to use an experimental design in which large number of software development teams (of the same size and balanced for skills and experience) are employed to develop the same software system, using the same software development methodology, for the same client, with half the teams randomly assigned to use an instantiation of the Ripple Implementation Framework and the other half not. By *controlling* all other factors that could potentially impact the quality of the system or performance of the teams in the experiment, one hopes that such an experiment would make it possible to establish cause-effect relations between any quality or performance indicators of the process employed by each team and the resulting product due to the use (or not) of the Ripple Implementation Framework. Also, if the number of teams is large enough, one could check for statistical significance in the measures (or indicators) of dependent variables as a causal outcome of using (or not) the Ripple Implementation Framework.

However, such an evaluation of a software development framework such as Ripple in an experimental setting with a large number of real teams of professionals building the same system using different development frameworks is practically not possible.

The most obvious reason is that such an undertaking would be ludicrously expensive given the resources necessary to conduct an experiment of such magnitude. In fact, we know of no real-world software development organization that can afford to do even *two* fully parallel development efforts on the same software product. Furthermore, given the complex nature of interactive-software development, it is impossible to control all factors (assuming it is possible to

identify all factors in the first place) involved in the development space. Furthermore, to show how difficult this would be in practice, here are some practical requirements for an organization as a setting for this kind of study:

- have the two life cycle processes (for SE and UE) in place,
- be willing to allow us to deploy a Ripple Implementation Instance in their development process,
- agree to let us observe and make measurements of their processes for the entire duration of the development process, and
- be able to afford multiple redundant development efforts for a given project.

Finally, even if the above problems could all be solved, any results from such an experiment will suffer from external validity problems as these results cannot be generalized to contexts where a different set of factors are at play: different size of development team, different skills or experience of teams, different software development methodology in use, different type of software system being developed, different type of client interaction, etc.

Thus due to the prohibitive cost of personnel, the impossibility of controlling all factors necessary to evaluate systems-level frameworks in a summative study, the lack of an organization that would allow us to mount such a rigorous study, and the infeasibility due to practical considerations, this kind of a study was not an option for evaluating the Ripple Implementation Framework. Therefore, we conducted an exploratory quasi-experiment, which is discussed in the next section.

6.1.2 Alternative kinds of studies and our approach

If controlled experiments are on one end of the spectrum of scientific rigor in evaluation, case studies are on the other end. Case studies often afford no control to the evaluators or observers on the various factors of the study. In this type of evaluation, one often observes a phenomenon and attempts to identify potential relations between the various aspects that were at play in the study and any perceived outcomes thereof. Because of the lack of control on the various aspects such as, say, team size in our case, these studies lack internal validity. In other words one lacks the power to ascertain the cause and effect relationship between the various aspects and perceived outcomes of interest in the study.

Another kind of studies which fall somewhere in between the two extremes described above are called quasi-experiments. Quasi-experiments are "experiments that have treatments, outcome measures, and experimental units, but do not use random assignment to create the comparisons from which treatment-caused change is inferred" (Cook and Campbell, 1979). Therefore, using the example of evaluating a software framework, these studies suffer from internal validity constructs and it is not entirely possible to relate the observed quality of the process and product to the type of framework used because, for example, different types or sizes of teams were used for the different conditions of software development.

Given the constraints stated above, we opted to conduct an exploratory quasi-experiment. The reason why we call it exploratory is because of the lack of control we had on different aspects of the study (discussed in Section 6.4) and the fact that we did not have enough number of teams to conduct a rigorous experiment (with potential for statistically significant results). The reason why we call it quasi-experiment is because we could not have randomized (and balanced) team assignments due to various constraints inherent in our approach (discussed in Sections 6.4.3 and 6.4.5).

We conducted this evaluation of the Ripple Implementation Framework in a classroom setting using the Ripple Implementation Instance described in Chapter 5. We compared eight groups of students, two of whom using and the others not using a Ripple Implementation Instance with measuring instruments described in later sections. Although we recognize that using courses as a setting and graduate students as practitioners has severe limitations concerning the fidelity to a real-world development environment, it was the only feasible venue for this study. Moreover, a classroom-based evaluation enabled us to address our two research goals (Section 1.5), i.e. to connect the SE and UE life cycles with respect to project development environments and by creating a cross-pollinated SE-UE course offering.

During the semester students in the UE class were trained for the usability engineer role and were taught the life cycle concepts and guidelines for building user interfaces. Similarly, students in the SE class were prepared for the software engineer role and were taught the life cycle concepts for functional core development. In semester-long team projects, these software and usability engineers worked in teams to develop software systems with interactive components for

a controlled set of real clients. Such a setting provided considerable control on the projects, team compositions and domain of expertise of team members, timelines, clients, project scope, evaluation, and observation. However, in this setting it was not possible to control factors that fell outside the purview of the classroom and the instructors such as team members' schedules, meeting habits, team members' expertise, etc.

We describe the experimental design, procedure, participants, metrics, analysis, issues, constraints, and limitations in the following sections.

6.2 Research Design

Our exploratory quasi-experimental study to evaluate a Ripple Implementation Instance in a classroom setting had two independent variables: domain of primary expertise (SE, UE, and SE+UE) and development conditions used by teams to match different real-world scenarios, as shown in Figure 22. The dependent variables were measures of communication factors among different life cycle roles, pedagogical factors, and others.

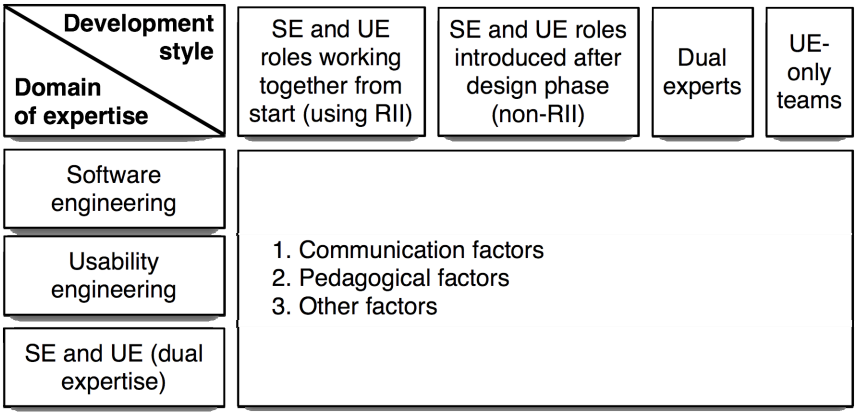


Figure 22: Experimental Design

The classroom setting allowed us some degree of control over the independent variables in the experiment, i.e. we were able to control distribution of students into teams. Domain of expertise, our first independent variable was determined as follows: students from the SE course were considered experts in development of the functional part of the system and students in the UE course were considered experts in development of the UI. Students taking both courses were considered as dual experts.

Development condition, our second independent variable, was controlled to a largest extent possible by assigning teams to different conditions. However due to an unequal number of students enrolled in the two classes, we had unbalanced number of teams for this variable. The team distribution and composition are shown in Table 4.

Team #	Team type	SE role	UE role	Role/setup/end product	Real-world scenario
A1	Joint team (Ripple)	3 SE students	3 UE students	• Distinct roles working together • Knew counterpart team from start • Developed fully-functional system	Teams with both SE and UE roles working together from start
A2	Joint team (Ripple)	3 SE students	4 UE students		
B1	Joint team (non-Ripple)	3 SE students	4 UE students	• Distinct roles working independently • Counterpart team introduced after design • Developed fully-functional system	Usability engineers brought in at the end of the life cycle
B2	Joint team (non-Ripple)	3 SE students	4 UE students		
C	Dual experts team	3 students who took both classes		• Played both roles • Developed fully-functional system	Software developers doing usability
D1	UE-only team	-	5 UE students	• Played usability engineer role only • Had no counterpart team • Developed high-fidelity prototype without functional core	UI work contracted to a different/external team
D2	UE-only team	-	5 UE students		
D3	UE-only team	-	4 UE students		

Table 4. Experimental setup and team distribution.

6.3 Procedure

Using the Virginia Tech joint course offering of Software Engineering (CS 5704) and Usability Engineering (CS/ISE 5714) described in the previous section, students from the UE class were trained in the usual way for the UE role and taught the life cycle concepts for building user interfaces. Similarly, students from the SE class were prepared, for the SE role and taught in the usual way the life cycle concepts for functional core development.

A semester-long project undertaken by each team constituted joint team project exercises crossing between the two classes and tying the two classes together. These projects required the students to apply concepts taught in each class by developing an interactive system (details of the system described in Section 6.3.2). The objective of the project was to simulate a real-world team- or role-oriented setting for the development of interactive systems.

6.3.1 Class setup and team distribution

As shown in Table 4, we had a total of eight teams. Five of these eight project teams included students from each class to represent a real-world scenario of software and usability engineers working as a team to build an interactive software system. One of these teams (Team C) was composed of dual-experts, i.e. students who were enrolled in both classes. These students had to play both roles as dictated by the project deliverables in each of the two courses. In the case of the rest of the four joint teams, each member played a distinct role: usability engineer if the student was enrolled in the UE class and software engineer if the student was enrolled in the SE class. Thus, each joint team had distinct SE and UE sub-teams.

In two teams (A1 and A2) the two sub-teams were introduced to one another at the start of the semester. These two teams represented an ideal condition where the SE and UE role communicated throughout the process (and using a manual instance of the RIF). They received help from the Ripple instance in the form of periodic emails to foster communication for synchronization, coordination, and constraint and dependency checks between the SE and UE processes and work products. Whereas, for the two other joint teams (B1 and B2), the sub-teams were not introduced to their counterpart teams until after the design phase was completed (two thirds of the way into the semester). This condition represented a common real-world scenario where usability engineers are introduced into the software development very late in the process. These two B teams (B1 and B2) did not receive any communication support from the Ripple instance. In all joint-teams, the UE sub-team developed high-fidelity prototypes of the UI and released them to the SE sub-team for implementation and integration into the functional core.

The three D teams were comprised of only UE students. These teams represented a scenario prevalent in some real-world situations where the UI design and development is contracted to a separate group or company. The UI specification from such a group is then implemented by a different team of software engineers (in house or elsewhere).

Students from SE and UE classes were asked to complete an SE and UE demographics questionnaire respectively on the first day of the semester (shown in Appendix A and Appendix B). These questionnaires addressed questions about educational and professional backgrounds, writing and software programming abilities, etc. of the students. Based on the responses on these

questionnaires, teams were created to ensure balanced skill-sets to the greatest extent possible. There was one exception to this team balancing attempt as discussed in Section 6.4.5.

6.3.2 Project objectives and deliverables

The team projects focused on developing an interactive system (or prototype in the case of D teams) including both user interface and backend functionality. Semester-long projects were used to build a real-world software system to facilitate the annual plant sale of the Horticulture Club of Virginia Tech. Three student members (two of them officers) of the Horticulture Club were trained to act as clients for all the teams. Special care was taken to prevent the client's interaction with one team from influencing their interaction with another team. The software platform, core requirements, amount of time spent interacting with the clients, and amount of information about the Horticulture Club were all pre-specified by us and all the same.

Each joint team (A, B and C teams) developed a fully functional software system to support the annual spring plant sale for the Horticulture Club. This plant sale entails placing bulk orders with a plant nursery, publicizing the event, managing inventory, and organizing the sale itself. Currently the Club members do not have any tool support to perform most of the activities involved in the plant sale. They do not have an efficient way to manage their inventories, know the plants that are in demand, see sales trends over years, have an online presence to facilitate requests by patrons for particular plants, and be able to make sales over the Internet. These were the high-level requirements that the Horticulture Club had for a potential software system. Therefore, the project topic for the student teams was to design and develop a fully functional software system to facilitate their annual plant sale. The three main components for this system included a web-based preorder system, an inventory control system, and a point-of-sale system. The three D (UE-only) teams developed a high-fidelity prototype addressing all envisioned functionality but without an actual functional core (the backend software).

The teams had a series of deliverables in each course throughout the semester. These deliverables corresponded with the major phases and products of any normal software development effort (including phases of interaction design development) such as the systems analysis, requirements specifications, software design specifications, usability specifications, etc. At the end of the semester the teams delivered the functioning system with code (in the case of

joint teams), high-fidelity user interface prototypes (in case of D teams), and other work artifacts created during the development process.

We required members from all teams to maintain a design and development journal to log all their meeting times, meeting minutes, activities undertaken in those meetings, composition of the team during the meeting, issues or problems encountered, etc. More information about journals is provided in Section 6.3.5.3.

6.3.3 Project clients

The client for this experiment was the Horticulture Club of Virginia Tech, a voluntary student-run body which hosts many community-oriented service events. One of the primary activities of this club is their annual plant sale, a three-day event that is hosted in late April each year. In the following sections we describe the role of clients and how they interacted with the teams.

6.3.3.1 Recruitment, remuneration, and client performance

We contacted the Horticulture Club with a proposal to have the student teams build a software system to help computerize the annual plant sale process. This proposal was met with interest and enthusiasm. We recruited three members from the Horticulture Club to act as clients, and paid a total of $1600 divided among them depending on the time and effort each of them expended over the semester. In total, the clients spent approximately 210 person hours during the span of the semester. The three clients performed exceptionally well and often went beyond the call of their duty in accommodating the teams' schedules and providing them with required background materials on the plant sale and the Horticulture Club. Their enthusiasm and dedication made the project experience better and the teams were excited at the prospect of working with a real client. Interacting with the clients motivated most of the teams to work hard and produce the best product so it would be selected by the clients for deployment. Unfortunately due to the tragic incidents of April 16 on the Virginia Tech campus we were not able to deploy the chosen system for use.

6.3.3.2 Risk, bias, and client role management

Initially we planned to have a different client group assigned to each pair of project teams. So, if there are a total of eight teams, this would translate to having four client groups. In such a setup, each client group would require at least two members to cover the role in case one of them drops

out of the study. Even with this kind of redundancy of client assignment, it would still be significantly risky in that, if both clients in any one client team drop out, it would be difficult to find another team of clients to take over that vacant position. Also, because of the fact that we will be distributing available budget resources ($1600) among all client teams, each client will receive only a small amount of compensation for their participation. This smaller compensation usually translates to smaller commitment and reliability. Therefore, in order to manage the amount of risk involved, we decided to have a single team of clients comprising of three members who can each be compensated more.

Next we faced the question of the optimum size of the client group. The Horticulture Club had many members, so we could spread the work over a large number of client representatives, assigning different members to different teams. This, however, would introduce a bias factor because not all members in the Horticulture Club were equally proficient with the full breadth of the plant sale operation. Only a few members of the club had overall knowledge about the sale and the various factors involved. The rest were usually assigned specific tasks pertaining to the sale such as putting up advertisements, attaching price labels, etc. So having client teams with club members without sufficient overall knowledge would have introduced inconsistencies and potential biases into the requirements and other client inputs. This meant we needed to have a smaller number of clients, with a focus on picking the most broadly knowledgeable from the Horticulture Club.

In order to manage these risks and bias factors, we decided to have a single client team of three. To prevent overloading this smaller number of clients, we had to carefully structure face-to-face interaction sessions, rather than allow unlimited direct contact (a realistic constraint in many real-world cases, anyway) with the teams. The clients were asked to give their available times and an equal number of equal-length client meeting time slots were provided to the teams. This way the clients' interaction and overall workload were controlled.

6.3.3.3 Training and control on interaction with teams

Every meeting with each team throughout the semester was also carefully controlled to ensure that there was no spillover of ideas or biases from one team to the other via the clients. The clients were informed about the different conditions in the study, and were instructed not to use

knowledge or insights gained via interaction with one team while working with the others. For example, before the start of the semester, the experimenter and the clients brainstormed to derive a baseline high-level requirement set for the plant sale system and the clients were asked not to deviate from that. The experimenter was present at every meeting the clients had with the teams and at times when the clients were not sure on how to answer a team's question without biasing the study, a discussion was held outside the meeting room and the clients were instructed on how to answer that question. Similarly after the initial requirements meetings with the teams where the clients described the baseline requirements for the system, the clients were instructed to give away additional information only when specifically asked by a team. For example, as part of evaluation plan the experimenter (with the help of previous year's officers from the Horticulture Club) prepared a package of different documents and artifacts from the previous plant sale and this package was given to those teams only if requested explicitly. A group email ID was created to include the three clients and the experimenter and all teams were given access to it to ask any questions or clarifications not resolved in these controlled face-to-face interactions. Once again, the experimenter ensured that any information given by the clients through this channel was not biased in any way.

6.3.4 Evaluation focus and specific hypotheses

In accordance to our research goals, this study had two high-level objectives:

1. Evaluate the effectiveness of a Ripple Implementation Framework instance that was created to connect the SE and UE life cycles and explore how and in what way the various communication factors impact the quality of product and process in an interactive-software development environment, and

2. Evaluate the effectiveness of cross-pollinated SE-UE curricula by exploring the various factors that impact the learnability of students in a joint SE-UE graduate course offering.

Based on these two high-level objectives, we had two over-arching hypotheses in this study:

1. Communication (among SE and UE roles) is the key factor that affects the quality of process and product in an interactive-software development effort, and

2. Students taking cross-pollinated SE-UE curricula have a better learning experience than those trained in SE-only or UE-only curricula.

More specifically, for this study the working hypotheses corresponding to these two overarching hypotheses were:

H1.a) The C team will create better quality software and have a better development experience than A teams because of the implicitly high level of communication and situational awareness afforded to a team with dual experts;

H1.b) The A teams will perform better than B teams because of the amount of communication, coordination, synchronization, and dependency checking afforded by the Ripple instance to A teams and the fact that they were introduced to their counterpart teams at the start of the semester;

H1.c) The B teams will perform better than D teams because B teams will have design ideas brought together from two domains, albeit late in the process and without any outside help from Ripple, whereas D teams will have a narrow user-interface-only design focus; and

H2) The students in this class will have a better learning experience as pertaining to learning the intricacies of interactive-software development than compared to traditional SE-only or UE-only curricula.

6.3.5 Evaluation metrics

As a part of developing the evaluation plan, we explored potential metrics to measure and compare the various variables in this study. In this study we tailored quality metrics to suit the constraints of evaluation in a classroom setting. We used product-based quality metrics such as complexity and conformity (to software requirements) for the SE component and various usability product metrics for the UE component. On the process side, we used measures of subjective opinions of developers from both life cycles and the actual amounts of communication, coordination, constraint and dependency checking, synchronization, and change management as supported by an instance of the Ripple Implementation Framework. In the following sections we discuss the background, our approach, and our metrics.

6.3.5.1 Background on evaluation metrics

Surveying the literature, one finds that software metrics are often categorized by various factors and methods used. For example, Herbsleb and Grinter (1998) report a case study where they used what they call "end-to-end" metrics which focus "on the overall software process." They contrast these metrics to more detailed ones that focus "on individual phases of development." Their so called metrics refer to function points to represent software size, effort in staff months to represent size (to indirectly show productivity), cycle time of development (from start of project to market release), and high severity defects identified in the software for the first half-year after release.

Another classification is provided by Basili, Briand, and Melo (1994) in which software metrics are either static or history. Static metrics are a measure of the complexity of software code at a given point in time whereas history metrics are supposed to provide a measure complexity over a given period of time. Another classification is provided by Sheppard (1988; 1992) where the quantitative aspects of software engineering are divided into code metrics (e.g., lines of code), design metrics (e.g., inter- and intra- module measures), and specification metrics (e.g., function points).

Cook, Votta, and Wolf (1994) present software metrics to analyze a development effort using data from past projects in an organization. In their case study they refer to three types of metrics: product-based, process-based, and process-behavior-based. They use the number of "source lines changed" and "source files changed" to represent product-based metrics; total time of the development process execution, "internal delay time," and the "developer who was involved" in the project to be process-based metrics; and the similarity of the development process execution to the process model to be process-behavior-based metric.

Similarly Fenton and Neil, in their widely cited paper (2000), provide a somewhat different classification of software metrics. According to them, software metrics are classified into product-based, process-based, and resource-based entities, each with two main attributes: internal and external. The product-based metrics cover aspects pertaining to specifications, designs, code, test data, etc. The process-based metrics cover constructing specifications, detailed designs, testing, etc. And finally the resource-based metrics deal with personnel, teams,

organizations, software, hardware, offices, etc. Each of these three main categories is stated to further have internal and external attributes. For example, the product-based metric associated to specifications has the internal attributes of size, reuse, modularity, redundancy, functionality, syntactic correctness, etc. and external attributes of comprehensibility, maintainability, etc. Similarly, the process-based metric associated with testing has the internal attributes of time, effort, number of coding faults found, etc. and the external attributes of cost, cost-effectiveness, stability, etc. Interestingly, the authors list "software" to be a resource-based metric with the internal attributes of price, size, etc. and the external attributes of usability, reliability, etc.

In addition, some software metrics are somewhat of a hybrid of the categories mentioned above (Duncan, 1988) and some use a widely different set of measures (Atkinson, et al., 1998). In spite of the variety of attempts to use, define, and develop metrics and measurement instruments as described above, there have been little to no evaluations of most of these existing methods (Emam, Moukheiber and Madhavji, 1993; Basili, Briand and Melo, 1996). Worse, there is little awareness about this area of software metrics in software engineering (Fenton and Neil, 2000). In our review of the literature, we found significant inconsistencies among the various classifications provided by various researchers and practitioners. The most used categorization that is somewhat consistent across most of the classifications seems to put software metrics into two broad types: product-based metrics and process metrics (Li, Dec 1999-Jan 2000).

Product-based metrics often try to determine the various aspects of a software system using the attributes of a product. Li defines these product-based metrics as the "measures of the software products" and cites source code and design documents as examples of this type of metrics (Li, Dec 1999-Jan 2000). One widely and historically used example for a product-based metric of source code is the total lines-of-code (LOC) of the software. This metric lost favor with the increasing diversity of the programming languages in the mid-1970s (the LOC of piece of software written in an assembly language could no longer be compared with that written in a higher-level language). This resulted in a new set of metrics that were a step closer to describing the complexity of the software. An example of this later time's product-based metrics is function points (with the assumption that functions are independent of the programming language) (Fenton and Neil, 2000).

With the advent of the object-oriented design paradigm and the dependence on inheritance, different set of product metrics emerged as the traditional attributes were no longer appropriate. For example, should one count the lines describing the attributes of a class towards LOC? Similarly, should one count the methods in a child-class towards function point counts if these methods are overwritten (Henderson-Sellers, 1999)? Examples of the new object-oriented product metrics included the depth of a class inheritance tree, number of children of a class (Chidamber and Kemerer, 1994), a class impact on superclass, class reusability, package size (Purao and Vaishnavi, 2003), etc.

We believe product metrics such as these are mostly valid across multiple projects (if at all) and are highly variable. Moreover, we believe these metrics are not directly representative of the quality of the product because of the lack of causality. For example, Fenton and Neil give an analogy and analyze the statement: "Data on car accidents in both the US and the UK reveal that January and February are the months when the fewest fatalities occur." They discuss how a naïve regression model based on this statement would predict that it is safest to drive when the weather is coldest and the roads are most treacherous. This conclusion is sensible based on the available data in the statement even though intuition reasons otherwise. The problem with this analysis is that the causality of this conclusion is weak to nonexistent. The authors argue that a more accurate analysis should take into account the facts that during these months there are fewer people driving and the fact that they drive much slower because of the inclement weather that is common in these periods (Fenton and Neil, 2000).

The authors argue that this driving analysis is analogous to the use of product-based metrics such as LOC or function points which appear to correlate but lack causality. For example a project with more function points does not necessarily indicate better quality than one with fewer function points even though they are created to solve the same problem. One should take a more rigorous approach towards measurement to have high causality between the measures and conclusions. Slightly higher-level metrics such as usability, reliability, maintainability, etc. are more representative of the quality of the software because these metrics are a composite of sub-attributes (for example, a usability metric includes quantitative metrics such as time on task, number of errors, etc. and qualitative metrics such as participant opinions). In either case, we acknowledge that in principle, the quality of a product is the true measure of the effectiveness of

the development process (Emam, Moukheiber and Madhavji, 1993). However, given the constraints of a semester and the available evaluation options, such a causally-valid product-based measurement was not possible.

The most widely used process metric in the literature appears to be the amount of resources expended during the development life cycle (Widmaier, Smidts and Huang, 2000; Li, Dec 1999-Jan 2000). One example of a process metric is the total number of person hours spent during the development effort (Duncan, 1988; Tate and Verner, 1991). Use of such a metric is possible in an industry setting because of the somewhat fixed hours of work per week. In an industry setting one can observe and record the number of hours a particular team spends on a project by using time cards for the hours clocked, etc. Even in professional settings where time-cards are not used, self-reporting would have checks and balances. For example, the total hours reported would have to be equal to the number of hours worked for the week. One cannot use a similar strategy in an academic setting because it is not possible to clock the students' hours. Each team potentially has six students with widely varying schedules and timetables. Students do not have fixed hours where they can be monitored to measure the amount of person hours they spent on the project. They may meet late in the evening to accommodate the different schedules of the team members. Self reporting of the hours spent by the students was the only option, and that was what we adopted.

Another important issue that prevents the use of these commonly used metrics as reported in the literature has to do with the need for steady state measures. Most of the product metrics are useful only when they are taken over time across multiple projects. If it were possible to offer the joint course twice, one with Ripple and one without, it would be possible to eliminate the confounding factors resulting due to the interactions the two sets of teams have in a single class. Also, the first time these measures are taken there is a high probability of confounds and biases because of the intrusion of the measuring instruments into the work activities of the project (Cook, Votta and Wolf, 1998). A steady state measurement was not possible in our evaluation because of the one time opportunity to offer joint SE and UE courses. Moreover, because of the learning setting the projects had some make-believe assumptions when it came to benchmark tasks, and performance specifications. This prevented any rigorous comparisons across products.

In light of the unsuitability of traditional software metrics for evaluating our work, we looked at other kinds of measures that can be adopted. One such measure is the level and amount of communication between team members as proposed by Bruegge and Dutoit (1997). In their work they provide evidence that "metrics on communication artifacts generated by groupware tools can be used to gain significant insight into the development process that produced them." In other words, the amount of communication could be a direct measure of the quality of the software being developed. The authors studied a set of projects and found that the electronic traffic generated by email and other groupware artifacts was representative of the level of overall communication in that project. Inspired by that work, we adopted a similar set of measures for this evaluation. Apart from the subjective reporting by the team members and the clients, we gathered objective measures of the amount of communication in the form of emails sent among the various conditions' team members indicating different types of communication and adopted those to be a direct representation of the quality of the software development process.

In order to accomplish this, we used an "empirically guided process" as described by Selby et al (1991), using a software process that included measuring instruments. An empirically guided process is based on the underlying principle that one should make measurement "active" by integrating measurement instruments into the development process. This contrasts with the passive use of measurement instruments where the measurements are obtained after the development effort is completed (Cook, Votta and Wolf, 1998). In our study we collected data from periodic deliverables each team submitted as per the course schedule and we required students to maintain artifacts such as journals as the semester progressed. We periodically checked these artifacts to make sure they were up to date.

6.3.5.2 Approach to deriving metrics

One cannot evaluate effectiveness of a framework such as Ripple using direct measures, but only in terms of indirect measures, effectiveness *indicators*, which are measures expressed in terms of other attributes. However, "utility is in the eye of the beholder" and one must choose a particular measure only if it helps understand "the underlying process or one of its resultant products" (Fenton and Pfleeger, 1997). Also, one needs to include both objective and subjective measures to gather a broad understanding of the process and product (Dix, et al., 2004). Fenton and Pfleeger go on to say that one can evaluate improvements in a process and its resulting products

only when the evaluation has a clearly defined set of goals in relation to the process and products, set within the Goal-Question-Metric (GQM) paradigm (Basili and Weiss, 1984; Basili and Rombach, 1988) for selecting and implementing metrics. In this paradigm one should first postulate the goals of evaluation and derive questions, the answers to which will indicate if the goals are met. Each of those questions is then analyzed in order to determine the measurements that are necessary to answer the question (Fenton and Pfleeger, 1997). Fenton and Pfleeger illustrate with an example of evaluating the effectiveness of a coding standard, which they show in their Figure 3.2, adapted here to Figure 23:

GOAL	**QUESTIONS**	**METRICS**
Coding standard effectiveness evaluation	User of standard	Who is using standard? -percentage of coders
		Coders experience with standard
	Productivity of coder	Coder's experience with standard
		Size of code -LOC -Depth of inheritance tree(in OO)
		Person hours expended
	Quality of code	Size of code -LOC -Depth of inheritance tree(in OO)
		Bug density

Figure 23: Example of deriving metrics from goals (adapted from Figure 3.2 in (Fenton and Pfleeger, 1997))

We used this approach to derive our metrics. Therefore, starting with goal one of this study, which pertained to the Ripple implementation instance:

Evaluate the effectiveness of the Ripple framework, as embodied in a specific implementation instance, to facilitate communication among developers within and between the two life cycles.

High-level questions, Q1 and Q2, can be derived from the goal:

Q1. *At the highest level, what are the indicators appropriate to reflect effectiveness of framework?*

Q2. *To what end and in what ways is communication used in the Ripple framework?*

Since an effective framework should result in better quality of process and resulting product, question **Q1** can be answered as:

A1. The quality of product and process.

And, since communication in Ripple is used to facilitate various other factors, question **Q2** can be answered as:

A2. Communication, the main contribution of the Ripple framework, is used to facilitate coordination, synchronization, constraint and dependency checking, and change management.

Now, one can ask, in questions **Q3** and **Q4**, what measures are needed to provide these answers:

Q3. *What are the potential product quality indicators?*

Q4. *What are the potential process quality indicators?*

Each of these questions can be answered with a list of indicators:

A3. *Usability (objective and subjective), completeness of requirements and design specification documents (objective and subjective), amount of unexpected changes (objective and subjective), number of defects (objective only), and code size (objective and subjective)*

and

A4. *Resources expended (objective and subjective), perceptions of developers (subjective only), how well the communication factors from **A2** above played a role (subjective only), and number and type of messages sent (objective only).*

Expanding A3 to specific measures, we get:

A3. *Usability:* ***time on task, satisfaction, number of errors***

Completeness (with respect to backend and UI coverage) of requirements and design specification documents: ***expert evaluation and developer perceptions***

Amount of changes: ***number of unplanned/unexpected changes as documented in journals maintained by students, developer perceptions on change handling and effects***

Number of defects: ***GTA evaluation of project reports, students' journal recordings of defects resulting from SE and UE interaction***

Code size: ***Lines of code, developer perceptions of size.***

Expanding **A4** to specific measures, we get:

A4*. Resources expended****: number of person hours recorded in student journals***

Developer perceptions of the process: ***as assessed by standard questionnaires***

Developer perceptions of effect of communication factors from ***A2****:* ***as assessed by standard questionnaires***

Number and type of messages sent between and among groups: ***analysis of email messages***

As a way of summarizing this discussion, the counterpart to the diagram of Figure 23 for the Ripple framework is shown in Figure 24.

However, certain changes and adjustments had to be made to these metrics as a result of the various constraints inherent in our study. For example, it was not possible to conduct a lab-based usability evaluation of all teams' products to arrive at measures such as time on task, number of errors, and satisfaction. The reason for this is that three D teams developed high-fidelity prototypes only and did not have a functional backend. Each of these prototypes were constructed to simulate carefully scripted benchmark tasks only and therefore it was not possible to run the same set of tasks in a lab-setting across all teams. Therefore we adopted a similar metric called overall product comparison (described in Section 6.3.5.3). Similarly it was not possible to compare code size and complexity using objective measure because the D teams had

no functional core. Also, in the five joint teams that created fully functional systems, they adopted different technologies ranging from AJAX to JSP, making it impossible to compare for code-based metrics.

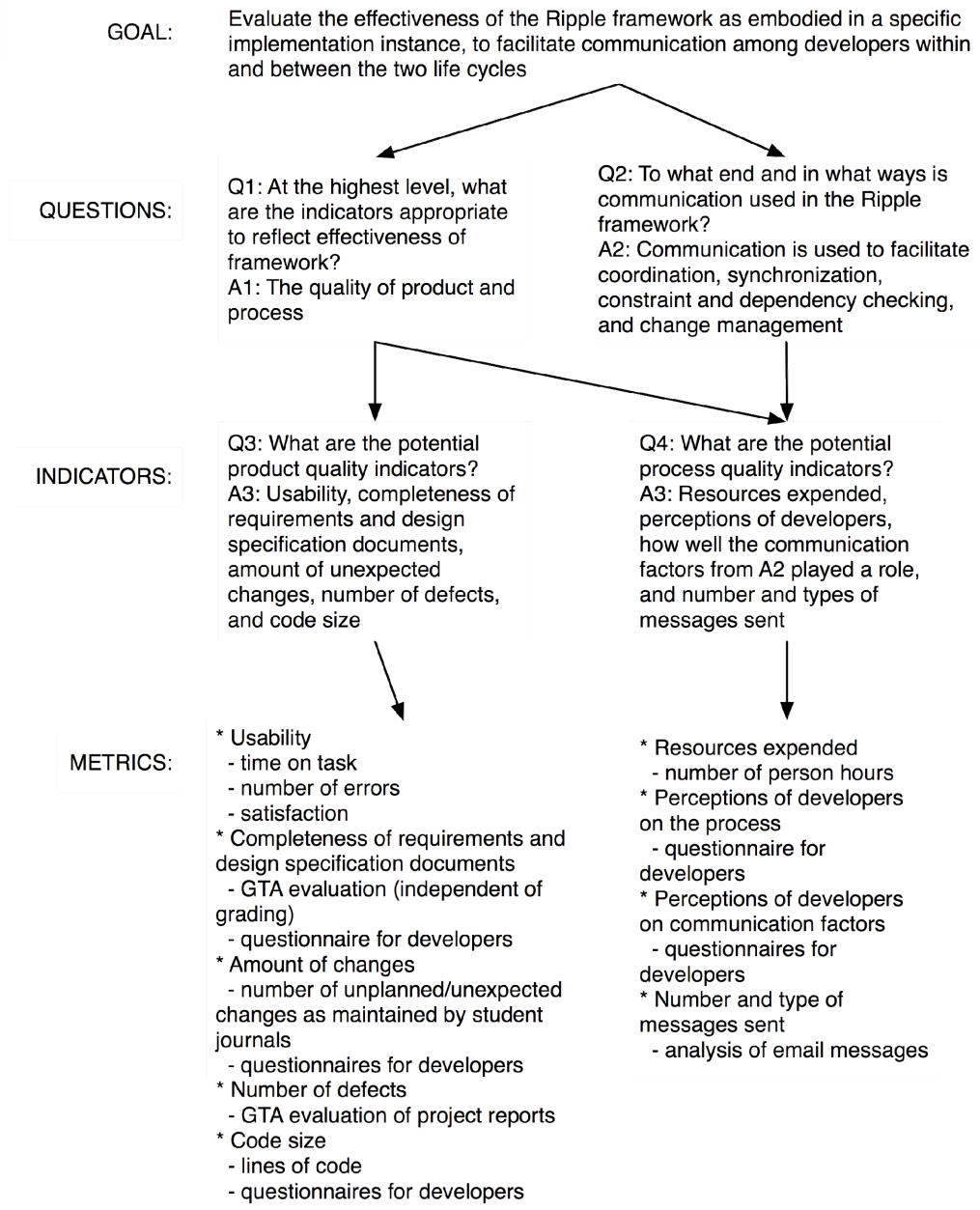


Figure 24: GQM framework applied to Ripple framework

6.3.5.3 Evaluation instruments and metrics

Using the approach described above, we derived the following evaluation instruments and associated metrics:

Overall product comparison

At the end of the semester, we conducted an eight-hour meeting in which, the experimenter and all three clients analyzed the prototypes for breadth of functionality covered by the UI, usability, and appropriateness for the Horticulture Club's goals. Each team's final system was analyzed in detail to arrive at an overall *value index* per team. First, we compiled an exhaustive "union" list of all features across all teams (e.g. ability to search by common name, ability to view shopping cart at all time, feature to provide directions to plant sale, etc.). Each of these features was then ranked on a *desirability scale*, which indicated how important this feature was for the clients' plant sale. The scale included a range of low, medium, high with fractional values in between in some cases. In other words this analysis provided a super set of all features with each feature's desirability present in all systems combined.

After this desirability analysis, the clients also rated each feature on each team's product on perceived quality. Starting from team A1, they analyzed each feature in each system and rated it on a *perceived quality scale* that included a range of poor, fair, good with fractional values in between in some cases. On a matrix, each feature on the union list was marked with a one or zero to indicate its presence or absence respectively, combined with the ratings for desirability and perceived quality. A product of these three values (feature present/absent, desirability rating, and perceived quality rating) was computed to arrive at a value index per feature. This value index per feature was aggregated to calculate the total value index per team project. Similarly, an aggregate feature count per team was also computed. These two values per team were treated as metrics for the study.

Journal analysis

As mentioned previously, as part of this study we required all students in both classes to maintain an online journal in the form of a spreadsheet to record individual and group hours spent working on the project, problems encountered, strategies used, negotiations held, overall impressions of the process, and other project-related details. This spreadsheet-journal was structured with the following sheets and fields:

1. Individual hours sheet
    a. date entry,

    b. number of hours spent working individually on the project, and
    c. description of the work done;
2. Group hours sheet
    a. date entry,
    b. number of group hours spent,
    c. the attendance at that group meeting, and
    d. description of the work done;
3. Unexpected changes sheet
    a. date in which an unexpected change, if any, occurred
    b. brief description of change,
    c. cause of change,
    d. any resolution reached, and
    e. additional comments;
4. Issues with SE-UE interaction sheet
    a. date in which an unexpected change, if any, occurred
    b. brief description of change,
    c. cause of change,
    d. any resolution reached, and
    e. additional comments;
5. Other experiences sheet
    a. date entry,
    b. experience, and
    c. any insight gained.

Using these journal entries, we computed the total individual and group hours per team. We used these as metrics for resources expended, and other data from the remaining sheets was used as subjective developer perceptions.

Email analysis

In order to measure the amount and nature of communication that transpired among the various members in each team, all teams (and in the case of joint teams each sub-team) were provided with custom group email IDs in which the experimenter was a silent member. The joint teams

each had three email IDs: one for the SE group, one for the UE group, and one for the combined group. The students were required to use these group email IDs for all project-related communication (and only for project-related communication) during the semester. The entire collection of email exchanged among all groups throughout the semester was archived and analyzed. Each email was tagged with various keywords (described in the next chapter). The total number of emails, the frequencies of keywords, and some subjective analysis were used as metrics.

Transcripts of end-of-semester symposium

As described in our pedagogical research objectives, it was essential for each team to share their perceptions of problems inherent in the development style they were assigned, the strategies they used to overcome these problems, lessons learned as part of their experience, and based on these experiences, any advice they have for real-world software developers. In order to facilitate this sharing of knowledge we hosted a research symposium at the end of the semester in which each team presented the key findings from their project experience. This symposium was recorded (audio and video) and the content transcribed (text files). Components of this data from the transcripts were used as metrics for the study.

Group interviews

At the end of the semester (after the symposium) each team was individually interviewed for an hour and a half. The interview was semi-structured with general questions probing the challenges faced, how prior work or academic experience in each role affected their performance in the study, how their interaction with the counterpart team affected their experience, what style of development they preferred, and other impromptu questions resulting from their answers. The interviews were recorded (audio only) and transcribed. Various parts of the data from these transcripts were used as metrics for the study.

Surveys

At the end of the group interviews, each student was administered two surveys: one with 55 questions gauging their perceptions on a wide variety of aspects of the study and another with 22 questions gauging their perceptions on various pedagogical aspects of the joint offering of SE and UE courses. The answers to these surveys were used as metrics for the study.

Subjective feedback from clients

After each interaction the clients had with a team (for example after their requirements analysis meetings, formative evaluation meetings, etc.) and after the end-of-semester product comparison meeting, the clients were asked if they had any general impressions on the team's performance in that meeting or their overall impression on the team's system. The feedback from clients in these situations were recorded and used as metrics.

6.4 Confounding factors and other issues

The academic setting chosen for evaluating our work had the potential for some inherent problems and issues. We list and discuss the salient ones here.

6.4.1 Conflict of ethical and research objectives

The key confounding factor in this study was the conflict between the teaching duties of the instructors of the two classes and scientific protocol to observe the teams without intervention. For example, when a team's performance (as perceived via the quality of work products they delivered) deteriorated, should we correct them or should we observe their performance without intervention? From the conception stage for this study, we have decided that the students' learning objectives would take precedence over any of the other factors. Therefore each team, in effect, followed a self-correcting process wherein they received constant feedback from the instructor and GTA on how to improve their performance. This was a significant confounding factor in the study, and worked against a divergence of quality across the team projects over time and a consequential loss of data about variations in performance across the teams.

6.4.2 Experimenter analyzing all data from the study

Due to budget, time, and the complex nature of the study, it was not possible to hire and train external coders to analyze data collected in the study. All data analyses in the study were conducted by the experimenter (except overall product comparison discussed in Section 7.2.1) and potentially subjected to evaluator biases. However, great care was taken to minimize this bias. For example, for email analysis (Section 7.2.3), the procedure was repeated twice to help ensure there were no errors. Similarly, for the group-hours portion of the journal analysis (Section 7.2.2), each student's entries were compared with the other members' entries to minimize reporting errors.

6.4.3 Unequal enrollment numbers in the two classes

Due to scheduling and other factors the two classes had an unequal number of enrollments. The SE class had an initial strength of 17 students, from which two students dropped the class in the first two weeks of the semester. The UE class, on the other hand had 33 students with no drops. Three of these students were enrolled in both classes. This variance in enrollment raised issues about the team compositions, forcing us to have UE teams with more numbers than their SE counterparts. This is somewhat in contradiction with the real-world development teams where usually SE numbers far exceed the UE roles. Also, we had just one team for the dual experts' condition due to the fact that there were only three members enrolled in both classes. This provided us an opportunity, albeit a statistically insignificant one, to see what role expertise in both domains played in influencing the dependent variables.

6.4.4 Experimenter being a GTA for one of the courses

Another potentially confounding factor is the fact that the experimenter was the GTA for the UE course. Since the experimenter has been a student of the CS department for a significant amount of time and many students knew about the work and the experiment prior to the evaluation, it was not possible to prevent the students from knowing about the experimenter's vested interest in the study. Therefore, it is possible that the Hawthorne effect (Parsons, 1974) was at play and it biased their ratings.

6.4.5 Team balancing issues

Due to the unexpected imbalance in the class enrollments of the two classes, not only did we have problems with team size as discussed above, we also had issues with team balance. Based on our past experience we anticipated about four students to drop from the UE class and therefore had a team of "reserve" students in one of the D teams who could each be reassigned based on other students dropping the class. However, no one in the class dropped, resulting in an unbalanced team (did not have equal skills for programming, prior background, etc.).

Another issue with team distribution stemmed from the broad range of prior background and experience in SE or UE students had from previous courses or industry experience in these areas. Specific cases where this experience played a role in the study will be discussed in the next chapter.

Also, since we were planning on evaluating the subjective perceptions of the students about aspects such as communication, coordination, synchronization, constraint and dependency checks, and anticipation and reaction to change, the relations between various team members potentially played an important role. It was important that the team members not cloud their ratings based on prior relationships and common ground with the other members of the teams. In other words, a student who has team members from the same research group or lab might have a different style of communication and interaction because of the common ground and social history. However, there were only a couple of teams which had such prior personal or working relationships and potential impacts on the study will be discussed in the next chapter.

6.4.6 Other issues

Apart from the above stated potential biases, there are other ones such as the fact that the students are working for a grade and will therefore go beyond any framework (Ripple or non-Ripple) to perform well on the projects. Also, this classroom project will be a finite, easily controllable, simulated project compared to the real-world projects where there is a possibility that the project might become intractable or go over budget and schedule. Another potential bias is due to the presence of measuring instruments that the students must use. As Cook, Votta, and Wolf states: measuring instrumentation in a software process "can be intrusive and expensive" and one encounters "resistance to the extra workload" because of the additional activities the developers should do to help in the measurement. This they say "often foils the study before it begins" (Cook, Votta and Wolf, 1998).

6.5 Constraints and Limitations

The joint course offering and project requirements imposed certain overheads for this evaluation. One such dominant issue was scheduling. Scheduling team meetings between the students across the two classes, as well as meeting the clients as a team imposed significant overheads on the part of the students. In addition to the scheduling overheads, there was an additional project phase for integration of the functional core and the UE for all five joint teams in this project. This extra burden is not usually present in the regular offerings of the SE and UE courses. Finally, we faced uncertainties in the measuring instruments employed in the study because, we did not have an opportunity to test them in the same scale and scope of use as were used in the evaluation prior to starting the study.

In light of all the potential biases and factors inherent in an evaluation in an academic setting, there were probably some factors that confounded internal validity. However, aiming for external validity, this setup was probably the best one possible in the given circumstances.

6.6 Summary of Evaluation Plan

The Ripple Implementation Instance was evaluated in a quasi-experimental classroom-based setting. Teams of students were formed from the SE and UE course sections to simulate the usability engineer and software engineer roles. The teams worked within the constraints of the semester and attempted to develop an interactive system to facilitate the annual plant sale for the Horticulture Club of Virginia Tech. The independent variables for the study included domain of expertise (SE, UE, and SE+UE) and development style. The dependent variables included factors of communication, factors for education in the context of a cross-pollinated SE-UE curriculum, and other project management factors. As part of this study, we required all students to maintain a journal to record individual and group hours spent working on the project, problems encountered, strategies used, negotiations held, overall impressions of the process, and other project-related details. All the teams and sub-teams were provided with custom group email IDs and were mandated to be used for all project-related communication. All this email from the entire semester was archived in a secure machine. We also had an end-of-semester joint SE-UE symposium where each team presented project experiences, lessons learned, advice to real-world developers, etc. After the symposium each team was individually interviewed for an hour and then administered a 55 question survey. The symposium and individual team interviews were recorded. Also, the instructors and GTAs graded each of the project phases and these grades were used as a secondary measure. Due to the academic setting and its inherent constraints and limitations, there was a possibility of significant biases and confounds. However, this was the best possible setup given the resources.

7 Chapter Seven: Analysis and Results

7.1 An Investigative Approach to Analysis

As described in Section 6.3.5.3, we adopted a series of measuring instruments to investigate the effectiveness of the Ripple Implementation Instance and to explore the various communication and pedagogical factors that could potentially impact interactive-software development. Each of these instruments provided a unique insight into the investigation of each team's performance and to the general understanding of the different factors that seem to influence the quality of interactive-software development. However, no single instrument, by itself, provided irrefutable evidence to explain beyond reasonable doubt the various factors that were at play in this exploratory study. Using an analogy of the criminal justice system, combinations of factors, ranging from objective to subjective and from qualitative to quantitative, needed to be pieced together to determine our understanding of what happened with each team in the study. In the process of reconstructing this picture, we identified some expected and some unexpected factors which could influence the quality of undertaking, or teaching about, an interactive-software development endeavor. However, these results should be read in light of the various confounding factors (discussed in Section 6.4) that could have influenced the study in one way or the other.

7.2 Analysis Procedures and Issues

In this section we elaborate on the procedure used with each of the measuring instruments to derive the associated data and the issues encountered in that regard.

7.2.1 Overall product comparison

As described in Section 6.3.5.3 the clients rated each feature in the product system from each team on a desirability scale (low, medium, high) and on the perceived quality of that feature (poor, fair, good). In case there were disagreements among the three clients about the appropriate rating for any feature, they were asked to discuss their disagreements and arrive at a consensus value for each feature. Figure 25 shows a sample of this analysis. The value of attribute "feature present" indicates whether that feature was present in each particular product system, was used accordingly as a binary "switch" to include a particular feature or not in the calculation of the value index. As an overall indicator of both desirability and perceived quality we used the

product of these two values and called it the value index of that feature. Because it is a product, a high value index requires both desirability and quality to be high. If either is rated low, the product will be correspondingly low.

					A1		A2		
				Desirability	Feature present	Perceived quality	Feature present	Perceived quality	Featu prese
Pre-order	Misc	Welcome screen		High	No		Yes	Good	Yes
		Directions to plant sale		High	No		No		Yes
		View Event Map		Low	No		No		Yes
		Make donation		Low	No		No		Yes
		Color coded categories		Medium-high	No		No		No
	Browse	All		High	No		Yes	Good	Yes
		Group of letters (of name)		Medium	No		Yes	Good	No
		First letter of name (from group)		High	No		Yes	Good	No
		Categories		High	No		Yes	Good	No
		View sorted	By name	Low	No		No		No
			By scientific name	Low	No		No		No
			By price	Low	No		No		No
			By category	Low	No		No		No
	Search	By all (full plant list)		High	Yes	Fair	Yes	Good	Yes
		By letter		Medium	Yes	Fair	No		No
		By price range		Low	Yes	Good	No		Yes
		By category		High	Yes	Good	No		Yes

Figure 25: Sample data from overall feature comparison analysis

The following numeric values were used to calculate the value index per feature (product of feature present, desirability of that feature, and perceived quality):

Feature present	Feature desirability	Perceived feature quality
Yes = 1 No = 0	High = 3 Medium = 2 Low = 1	Good = 3 Fair = 2 Poor = 1

Table 5: Numeric values used to calculate value index

One of the issues faced in comparing the value of the systems developed by the different teams in the study was their dissimilarity in terms of the "realness" of the systems. For example, the three D teams created systems without a real backend (i.e. user interface only). Therefore, the value index for D team systems are more perceived quality attributes than real ones. However, we believe this value index still serves as a basis of comparison on a per-role system quality, i.e., within sets of teams that used the same development condition. For example, the non-D teams had joint teams (or had two trained roles in the case of C team) to create a system with two components: UI and backend. The D team had only the UE role and created a UI-only prototype.

Another issue with this instrument is the fact that it does not provide an insight into the contribution of each sub-team in the case of A, B, and C teams. For example, a low value index indicates a failure on the part of the entire team but does not describe if the failure was due to a low-quality contributed by the UE sub-team or the SE sub-team.

Similarly, another potential problem with this index is the fact that it is not tolerant to distortion due to feature creep. For example a team could accrue a high value index if its system has fewer core (essential) requirements but a large number of non-essential requirements, which would probably never be used by the client. This problem has the potential to be more pronounced in the second metric derived from this instrument: feature count per system. However, to our knowledge, this was not a severe problem in this analysis.

7.2.2 Journal analysis

As described in Section 6.3.5.3 all the students in the study were required to maintain an online journal in the form of an electronic spreadsheet that was periodically checked by the experimenter. However, certain students were not good at keeping their journals up-to-date in spite of repeated reminders. Worse, it appears that some students concocted fictitious meeting times and hours just to adhere to the requirements of keeping the journal. There was also some resentment as evident in their journal entries about the extra work required to maintain the journal. These aspects are described in later sections of this chapter. Here we focus on the procedure used to minimize the reporting error to the greatest extent possible.

Of all the entries in the spreadsheet (individual hours spent, group hours spent, unexpected changes, issues with SE-UE interaction, and other experiences), the measure that was most susceptible to problems of incorrect or incomplete reporting were the person hours spent working on the project. This is because inaccuracies in reporting of this data, even by one or two members in a team, leads to an inaccuracy in the entire team's data. Whereas, not reporting unexpected changes or other experiences, at worst, fails to provide a full picture of what happened, but does not distort the data of the entire team. Therefore a lot of effort was put into making sure the effects of errors with respect to hours worked as reported in the journals were minimized.

There was not much that could be done with the reporting of individual hours because those data points could not be correlated with any other data points. However, for the group hours, it was possible to reconstruct the exact meeting times, duration, and attendance by piecing together the entries from all the team members' journals. For example, if a meeting was reported to have taken place at a certain time for a certain length and attended by a certain number of team members, it was possible to detect differences (and often to deduce the correct values) by looking at each of the reporting member's entries. The meeting times were also cross-checked with their email exchanges in some cases to ensure their accuracy. Using such thorough, but often tedious, reconstruction it was possible to ascertain, with a reasonable degree of certainty, the exact number of people who attended a group meeting and the duration of the meeting. By reconstructing the teams' group meeting hours it was possible to compute the total person-hours spent by each sub-team and joint-team for the entire duration of the semester. Also, using this reconstruction, it was possible to exclude hours spent working on non-project related activities (for example, working on homework assignments that were not part of the group projects).

As for the qualitative data in the journals (unexpected changes, issues with SE-UE interaction, and other experiences), we analyzed each entry in each developer's sheet and used that information as evidence to investigate and reconstruct what happened in this exploratory study. This instrument proved to be an important source of information in the investigation of how the teams performed and aided in identifying a list of factors that seem to play a role in the interactive-software development space.

7.2.3 Email analysis

As described in Section 6.3.5.3 all the students in the study were required to use, for all communication about the project, group email IDs provided to them by the experimenter. Each A and B team were assigned three group IDs: one for the SE sub-team, one for UE sub-team, and one for the combined SE-UE team. The C and D teams were each assigned one group ID. The experimenter was a silent member in all the groups and archived all messages sent via all group IDs. There were a total of about 5300 emails that were sent over the semester by all the teams. For analysis, these emails were grouped by team first and then by component sub-teams in the case of A and B teams. The experimenter then analyzed each of these emails and tagged each with a keyword per unit of communication. We define a unit of communication as one complete

idea that was expressed in a message. For example, in a message informing that there is need to discuss project deliverable two and that the sender is available to meet on Friday at 3.00PM, there are two units of communication: one unit “planning and execution” and one unit of “scheduling”.

The Mail program on a Macintosh computer was used to archive the emails. A tagging plug-in called MailTags 2.0 (MailTags, 2007) was used to tag the emails. Because this tagging plug-in does not allow for multiple occurrences of a particular keyword for the same message, multiple keywords representing occurrence counts were created. For example, for the keyword “planning and execution”, 13 instances were created (1- planning and execution, 2- planning and execution, 3- planning and execution, etc.) each representing the frequency of this keyword’s occurrence in a message. This tagging exercise was conducted twice over the entire archive of email to account for any errors or omissions.

Apart from the quantitative data such as frequency counts and amount of communication as indicated by the aggregate of communication units across each team, the experimenter recorded subjective observations of issues arising in the email content during the tagging process. These were also used in the analysis process. This instrument proved to be an important source of information in the investigation of how the teams performed and aided in identifying a list of factors that seem to play a role in the interactive-software development space.

The keywords used in this tagging exercise are described here:

7.2.3.1 Ripple

This keyword was used to tag communication units related to the Ripple instance in the study. Specifically any of the following types of messages were tagged with this keyword:

- Messages about managing the team's work product repository in the form of "Deliverables" thread in the group email IDs,
- Messages about scheduling in the form of "Posting project activity plans" thread (messages used to facilitate constraint messages for A teams), and
- Messages with introductions to counterpart teams by experimenter

7.2.3.2 Planning and execution

This keyword was used to tag all messages containing any of the following type of communication units:

- Messages with introductions and exchange of contact information among team members at the start of the project and after the counterpart teams were introduced
- Messages about project planning that are relevant for the functioning of the team and progress of the project
- Communication with and about client relating to the project
- Preparation/planning for project deliverables (and project document attachments)
- Messages related to work product exchange among the group
- Discussion among group about incompatibilities with what was done by other members
- Requesting for feedback on work products
- Discussion of technical skills or lack thereof required for project
- Questions (and answers) about project scope directed to the experimenter (e.g. “should we consider credit card accounts?”, “are we doing this right?”)
- Roles and project task division and assignment
- Messages in which a previously mentioned or discussed project-related proposal or idea was acknowledged or agreed upon
- Messages in which ideas related to project planning, execution, etc. (but not scheduling locations, times, etc., which was included in scheduling keyword described in Section 7.2.3.6) were put forward
- Messages in which a previous project-related proposal/idea was disagreed upon

- Messages in which there was further discussion about agreement or disagreement about a proposal/idea

- Messages in which there was a follow-up to a question without agreement or disagreement (e.g. I don't think we decided the meeting place yet)

7.2.3.3 Strategies

This keyword was used to tag communication units related to strategies for and approaches to structuring, planning, and about adopting creative techniques for project execution (e.g. adopting separate email threads for different types of project communication, creating a group calendar to ease scheduling problems, etc.)

7.2.3.4 General communication

This keyword was used to tag instances of messages dealing with communication other than that related to the project. Specifically:

- Technology discussions/advice to team members (e.g. "Microsoft Visio is better than Word for images")

- Personal issues of team members not related to project (e.g. "I will be late to the meeting because I need to pick up my kids from soccer practice")

- Communication slips (e.g. "Oops! forgot the attachment, here it is")

- All other issues that have an indirect effect on the project and team but not specifically related to the content of the project (e.g. "I will be going to a conference next month so I need to get my part of project deliverable before I leave").

7.2.3.5 Administration

This keyword was used to tag communication units related to class administration and management (e.g. project due dates changed, homework submission instructions)

7.2.3.6 Scheduling

This keyword was used to tag instances of messages dealing with scheduling and logistics issues. Specifically:

- Communication about, and negotiation of, planning/availability for team/client/other meetings
- Requests for confirmation of meeting times and places
- Experimenter responses (where necessary) to these messages

7.2.3.7 Deliverables

This keyword was used to tag instances of communication units related to the mechanics of upcoming deliverables but not about the project content itself. Specifically:

- Discussions on formatting, binder requirements, other logistics for deliverables
- Questions and responses (from team members and experimenter) about these issues
- Messages submitting deliverables

7.2.3.8 Emoticons

This keyword was used to tag messages with emoticons such as smiley faces, phrases, humor, friendly salutations, greetings, etc. (e.g. “hope you all had a good weekend”, “I want to punch someone”)

7.2.3.9 Affirmations

This keyword was used to tag short acknowledgments to questions/answers (e.g. “got it”, “thanks”)

7.2.3.10 Team cohesion

This keyword was used to tag instances of messages illustrating the formulation of, or existence of, good team relationships or cohesion (e.g. bonding, team unification). Specifically:

- Complimenting the team or team member's work, ideas, etc.
- Giving heads-up to team about future personal trips, leave of absence, other issues that might affect team performance, and how they will compensate for these issues
- Volunteering for help other team members after one’s assigned work was done
- Providing feedback such as "sorry it took so long" and other related messages

- Wishing other team members good luck about outside engagements (e.g. “good luck on your job interview”)

7.2.3.11 In-class activity

This keyword was used to tag messages pertaining to in-class activities and homework assignments not part of the team project.

7.2.3.12 Credit card authorization and testing

Messages related to account setup and testing of the third party credit card authorization client (Click & Pledge, 2007) that was setup by the experimenter for this study.

7.2.3.13 Counterpart-role specific

This keyword was used to tag messages where the interaction with counterpart role was discussed (applied only to A and B teams). Specifically:

- Asking/assigning/delegating jobs to the counterpart role
- Preparing, anticipating, and planning for interaction with counterpart role

7.2.3.14 UE initiation

This keyword was used to tag messages that were initiated by the UE team to the SE role (applied only to A and B teams).

7.2.3.15 SE initiation

This keyword was used to tag messages that were initiated by the SE team to the UE role (applied only to A and B teams)

7.2.3.16 SE-UE interaction

This keyword was used to tag messages indicating communication between the two counterpart roles. Specifically:

- Scheduling and introductions between the two roles
- Discussion of SE/UE issues
- Team C's discussion about counterpart role

7.2.3.17 Inter-role negotiation

This keyword was used to tag messages between the two roles related to issues, compromises, and discussion related to working with the counterpart role. Specifically:

- Questions about feasibility and other issues between SE and UE roles
- SE/UE people responding with issues about other role
- Questions about what is needed by the other role and answers
- Discussion of timelines and schedules and constraints of one role on the other.

7.2.3.18 Team issues

This keyword was used by experimenter to tag instances of messages indicating problems with teams. Specifically:

- Lack of response to emails or certain team members not pulling their weight
- Miscommunications resulting in negative messages
- Tone indicating lack of team cohesion

7.2.3.19 Repeat communication

This keyword was used to tag messages where communication that was previously discussed was repeated because of lack of response or action from other team members.

7.2.3.20 Social

This keyword was used to tag messages where the team discussed social activities not related to project or other class related issues.

7.2.3.21 Miscellaneous

This keyword was used to tag messages related to all other issues not covered by the keywords above such as issues pertaining to re-sizing of teams because of students dropping or intervention of instructors and experimenter to fix team problems.

7.2.4 End-of-semester symposium

As described in Section 6.3.5.3 each team was required to share their project experiences with the class. This symposium, which lasted for four hours, was recorded (audio and video) and the content transcribed (text files). Even though this transcription exercise took significant effort, it provided rich data about what worked for each team and what challenges they faced. This instrument provided an overview of the entire project as perceived by the teams.

7.2.5 Group interviews

As discussed in Section 6.3.5.3 each team was interviewed for an hour and a half at the end of the semester. These semi-structured interviews were focused on probing specific aspects related to each team and for following up on certain topics that teams brought up during the symposium. The audio of these interviews was recorded and then transcribed. Once again, this was a tedious and time consuming exercise, but one which yielded rich qualitative data about many aspects of each team's experiences.

7.2.6 Surveys

As mentioned in Section 6.3.5.3 at the end of the group interviews the experimenter administered two surveys to each team member: one with 55 questions gauging their perceptions on a wide variety of aspects of the study and another with 22 questions gauging their perceptions on various pedagogical aspects of the joint offering of SE and UE courses. Due to IRB restrictions, we were not able to collect any information identifying the team members in these surveys. Therefore individual level comparisons were not possible across surveys and email data. Thus, the surveys were aggregated to team level responses and analyzed. Also, all students were given the same two surveys regardless of the development condition they used for their group project. Therefore, for questions aimed at gauging their perceptions about other development conditions, the students had to resort to conjecture based on their experience observing the other teams throughout the semester and during the symposium presentation. For example, some questions in the survey asked the students about their perceptions on the importance of having periodic communication between SE and UE roles in an interactive-software development effort. However, the D teams had no first hand knowledge with this issue as they did not have a counterpart SE role.

7.2.7 Client feedback

As discussed in Section 6.3.5.3 after each interaction with the teams, the clients were asked to provide general impressions on the team's performance and their system. These were purely subjective assessments by the clients that provided a perspective that was not captured in any of the other instruments described above. Whereas each of the other instruments (except, perhaps, the symposium) provided unique insights into a small subset of the aspects involved in this study, this client feedback provided a holistic, albeit general, assessment of each team. Their feedback represented those aspects that are more about the overall interaction and impression they had about each team and its system.

7.2.8 Experimenter observations

Throughout the study, the experimenter kept notes about the different teams, interesting events, and other observations. These observations, for the most part were in accordance with those of the clients.

7.2.9 Team level analysis

Except email frequencies and person hours from journals, no quantitative metric could be derived at an individual level. Therefore, most of the quantitative analysis had to be performed at a team-level. Since we had only eight teams in all, with two teams in A condition, two in B condition, one in C condition, and three in D condition, the study did not lend well to finding any statistically significant trends among the different team types. Also, as will be described in the following sections, one team from A condition and one from B condition were anomalies and therefore skewed the quantitative results even further. Therefore most of the analysis in the following sections relied on qualitative data with an exception of survey data.

7.3 Hypotheses and Data Analyses

In this section we take each of the hypotheses and present relevant data obtained from the various instruments discussed above in conjunction to them. We state each of the hypotheses from Section 6.3.4 here and discuss related data:

7.3.1 Hypothesis H1.a

The C team will create better quality software and have a better development experience than A teams because of the implicitly high level of communication and situational awareness afforded to a team with dual experts – ***not verified***

We did not find any data to support this hypothesis. On the contrary, we found various insights that appear to indicate that developing interactive software using dual experts may actually result in a poor quality process experience and resulting products. In the following sections we use different data items and potential factors that explain why this hypothesis was not verified.

Using the two measures of system value index and total feature count from the overall comparison instrument, Team C came seventh out of the eight for both measures (we show this in Figure 27 and Figure 26 as a descriptive instrument, and not for statistical inference).

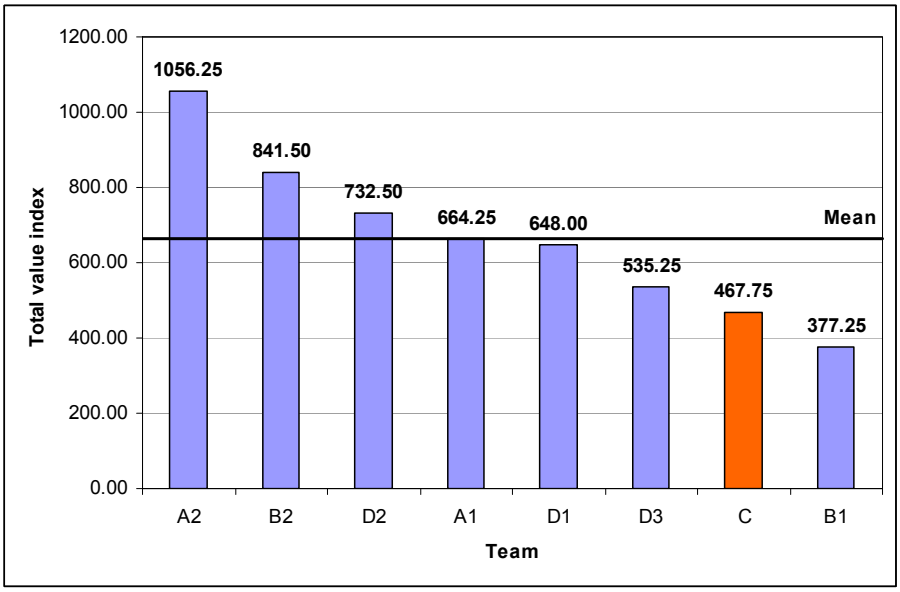


Figure 26: Total value index of all teams, showing relative standing of Team C

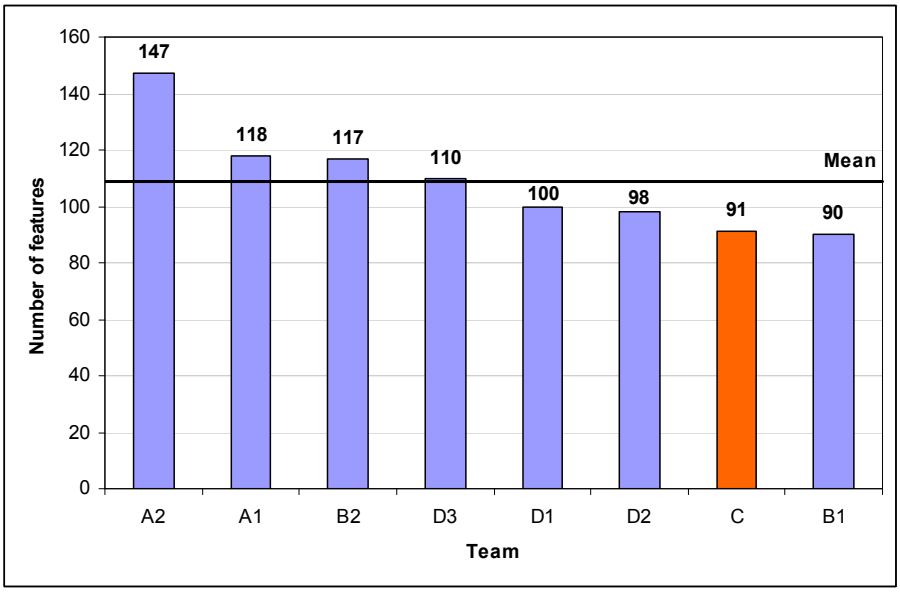


Figure 27: Total number of features per team, showing relative standing of Team C

However, because of the single data point in this condition (only Team C), it is impossible to say if this is truly representative of this condition or an anomaly. For example, if there is another C team that performed exceptionally well, the range (the height of the box) as shown in Figure 28, would provide a different insight into the C condition because it would result in a tall box showing no difference with the other conditions.

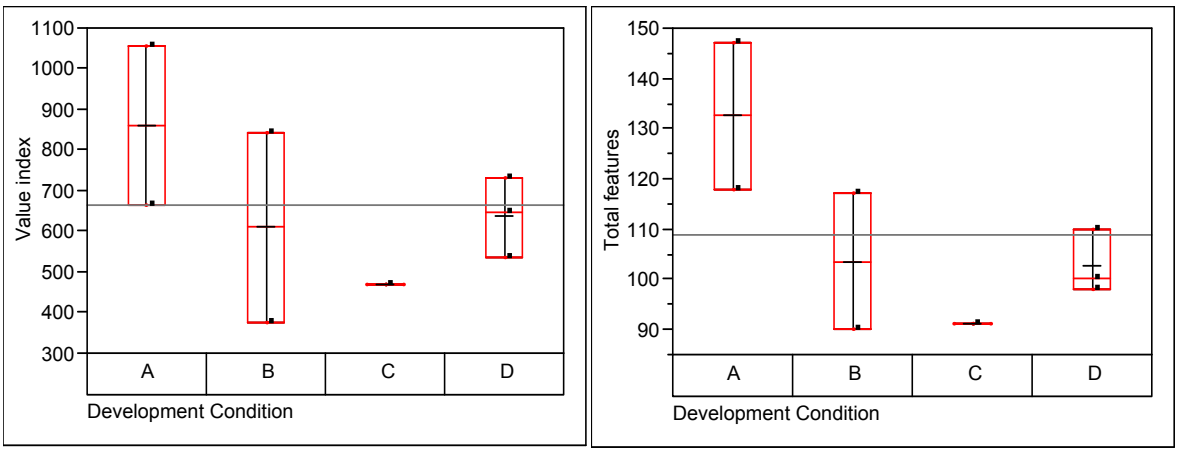


Figure 28: Condition-level comparison by range for value index and total features

In the following sections we analyze some factors that potentially contributed to the poor performance of Team C.

7.3.1.1 Team size, balancing, and dynamics

Team C had substantial issues with intra-group dynamics and cohesiveness. We describe these issues here:

Unbalanced team

As discussed in Section 6.3.1, we used demographic questionnaires (Appendix A and Appendix B) to balance teams for comparable experience in SE, UE, real-world experience, writing skills, etc. Such balancing was not possible for Team C because we only had three students registered in both classes and they had to be placed in a single team. This imbalance could have contributed to this team's poor performance.

Team problems and experimenter intervention

Every semester when we teach the UE or SE class we administer surveys to students periodically during the semester to detect team problems. During this study, we had journals that provided a deeper and more insightful understanding of each team's performance and problems. Team C happened to be one team that had many problems. There were issues with certain members not performing their share of the work and unprofessional behavior (e.g. not responding to emails in a timely manner, not returning certain work products in a timely manner for integration into project deliverables, etc.). These factors could have influenced this team's performance. Given that the educational goals of the class had to take precedence over the study goals, the experimenter had to intervene and admonish the problem members of the team. Although this intervention surely dampened this effect, nonetheless, some problems persisted.

Lack of equal participation by team members

Related to the team problems described above, we also found that the individual hours spent by the three members for this project to be widely differing, as shown in Figure 29. This disparity in team-member effort probably explains why the team had problems, which in turn almost certainly influenced their performance.

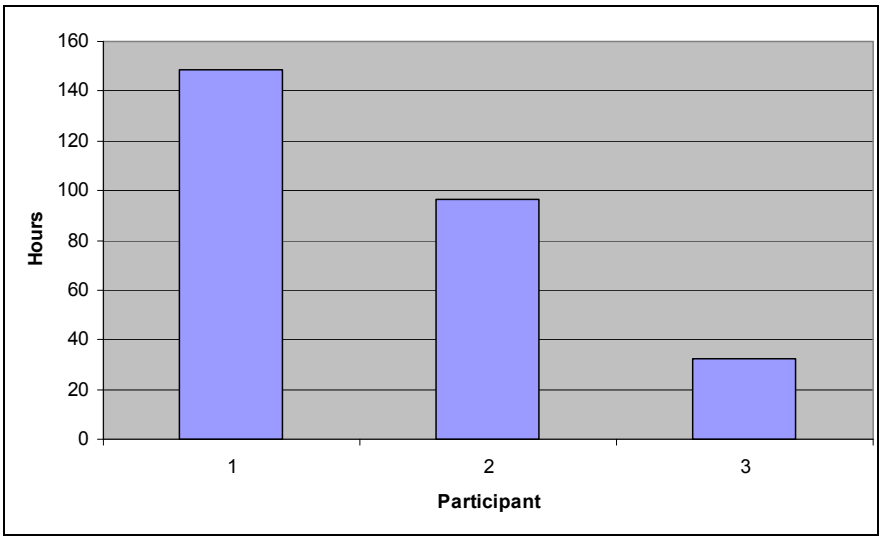


Figure 29: Team C: Individual person-hours reported as worked, totaled from individual journals

Team size and work load

Due to the fact that only three students enrolled in both classes, Team C's size of three was a result of a constraint rather than by design. When asked during the group interviews if a fourth member would have helped them, they said yes, "because in other teams they had four members working on a UE deliverable and three for the SE side… but we were just three for both", comparing themselves to A and B teams (D teams had about 5 members per team). This small size could have contributed to lower than expected performance as they were constrained to ideas and creativity of three people instead of the larger groups in other conditions. However, as discussed later, the Team C developers all reported that their work from the SE and UE classes overlapped about 40-60% thereby potentially compensating for the smaller team size. There is no evidence seen in any measuring instrument used in this experiment to verify or contradict such compensation.

7.3.1.2 Potential disadvantages of the C condition

Cognitive dissonance between the SE and UE roles

Given that the members of Team C had to play two roles in this project that are often at odds with one another, they could have been a victim of cognitive dissonance. As remarked by my team member in group interviews: "we had to enter [the SE and UE tasks] with a different perspective… with a different terminology and such." However, this remark is in contradiction to what they said about how the fact that they had to play both roles helped in reinforcing their

designs and in reassuring them that they were on the right track (see "Reassurance of being on the right track" below).

Inherent conflict of interest between the two roles

Given that the members in development condition such as that of Team C had to play both SE and UE roles in the project, it appears that they were subjected to an inherent conflict of interest. In the words of a team member: "… we had no pure perspective. We couldn't tackle UE problems with a 100% UE perspective, same thing for SE [..] which wasn't intended but it couldn't be helped". They acknowledged that their system was just "sufficient and […] minimalist" and that they could "have used some conflicting opinions". During the group interview, this aspect was probed further and we reproduce that part of the transcript here:

> *Team C member: We always knew what the other [role] was going to [do] so it kind of limited what we were doing. We made sure that the basic requirements that the clients wanted were captured in the system but we did not go out of the way to develop something… or make extra features done or something like that.*
>
> *Experimenter: Because you had to implement them yourself?*
>
> *Team C member: Yeah. We couldn't be like the other teams where they said, what the heck it is not our problem… it is the problem of the SE team. We were the SE team as well!*
>
> *Experimenter: What were some examples of features which you left out because of that?*
>
> *Team C member: Search option. Search option in preorder website.*

Also, Team C was the only one in the study that developed a standalone component of the annual plant sale management system. This was not appropriate to the client's workflow because they do not have dedicated computers to have a standalone system installed during the plant sale. All other teams opted for more appropriate, but much harder to implement, web-based solutions. When asked about this in the interview, they said they had no real reason for doing this, implying they chose the easier of the two options.

Effective software development entails a fine balance between a creative design endeavor and a practical implementation effort. In the case of dual experts, there appears to be an imbalance

between the two with the developers choosing the easier way out of design tradeoffs. As one of the Team C members remarked “well it would have been nice if I don’t have to code [the UE design]” implying that they would have emphasized the design more if it did not entail more work. They also commented that an independent review of their work by a dedicated SE or UE role would probably have helped them create a better product.

Life cycle constraints

Because of the classroom setting, and the way deliverables in the two classes were scheduled, Team C appears to have had tougher constraints with respect to functioning as dual experts. Whereas in other joint teams there were two distinct sub-teams working on the UE and SE deliverables, Team C had three people to work on them in the same amount of time. One of the Team C members commented “time constraints also became a problem… for the other team they had two weeks each but we just had one two week [period] for us. Of course we do agree that there was a lot of overlap but we can't just use the [same] deliverable right?”

Also, as reported by one member in this team: “[when] it comes into the actual coding phase the design [got] to be more sophisticated for the UE side. [This was] a problem. As we started coding, we had to draw from the initial prototype in UE and this was kind of awkward because we were also testing the UE prototype at the same time. So [as] that was [be]coming better, we were finding problems with that, but we couldn’t really change that in our design through the code itself. We just had to keep going on that. The modules itself, we coded them, tested them, and brought together. That was fine. There was no real integration phase in the sense that the other groups faced. This would have been around the part where the other groups combined the UE's design with the software engineers’ code. We didn’t really have that so that was kind of a plus judging from some of the comments from the other groups.” It appears that the problem of implicit differences in iterativeness between the SE and UE life cycles was compounded in the case of dual experts by the fact that both roles are played by the same people. Quoting a Team C member “it's tough to program the interface side of the project when we haven't done all the testing for it yet, but we need to have something in there for the application. We have to program the app, but the interface testing isn't finished [yet]. […] We were kind of working with a system that we knew was flawed, but just kind of had to go with it”; the urgency to finish a work product for the other role’s use seems to be less pronounced in the case of dual experts because

they are the other role as well and therefore had no external pressure to complete and share the work product. On the other hand, one could also argue that this is actually an advantage because they could have planned better to facilitate their other role's work activity and that there was no need to negotiate with a counterpart role as in the case of other joint conditions. However, the extra diligence required to pursue this advantage often is lost in the scheduling and other pressures of a development project.

7.3.1.3 Potential advantages of the C condition

Overlap of work activities and reduced workload

One of the advantages of being in a condition where the same role is performing work activities from both life cycles is the amount of overlap between the two life cycle processes. During the symposium Team C reported that 50-60% of their work overlapped. This was in accordance to the data from the survey instrument where all three members of this team chose 40-60% for the question asking about the amount of work overlap between the two life cycles. As one of the Team C members said "because they [SE and UE processes] are very related, […] have similar objectives for each, [have] work products [that are] very similar if not almost identical in some cases, and many times one side's work in a phase influenced the other, [… the] advantages for us in the sense that we were students in [both] classes and [that] there was a lot of overlapping information, […] while we were doing a lot of these documents, a lot of the information could be reused or rewritten, drawn from a lot of research [that] was behind these things that were already done. Some examples [were] product concept statement and the product overview document, the hierarchical task analysis and the data flow diagram, use cases and scenarios. They all wove themselves together fairly well". Numerous other comments supported this apparent overlap. Here are a few examples:

"While developing the product concept statement and the product overview document, we found that the two overlapped. The technical description that we [provided] with the product concept statement [as part of deliverable for] UE served as the outline for the product overview document that we had submitted for SE"

"Then for the SE deliverable, we created the SRS and part of this document had use cases which describe possible task flow, work flow through the system […]. These are very close to what UE

uses scenarios [for] so we kept these and would rewrite these for an upcoming UE deliverable. Also in the SRS [there] was the dataflow diagram. This also showed the potential workflow through the system and different levels of modularity in functionality. This was very similar to the UE [life cycle's] hierarchical task analysis which shows a task […] broken down to subtasks [...]"

"We found that as we were [preparing] the low-level design document, we were taking UE concepts into mind. We used our screen designs that we had made to think about what other [software] classes we might need"

"Immediate collaboration"

Another advantage with a dual expert condition is the prevalence of, as one team member put it, "immediate collaboration". There are no operating and scheduling overheads for working with the counterpart role; a significant issue in the case of other joint team conditions. In the words of one of the Team C members: "some advantages to being dual experts is that there is immediate collaboration, so really if we needed something we didn't really had to go to another group or setup a meeting. So [we] kind of knew everything that was going on. So this also lends itself to having complete contribution and control. So the design was always consistent because we always knew what was [going] be there. When it came for integrating the functionality and the interface, there were really no surprises because we planned it the whole time. That was kind of nice."

Reassurance of being on the right track

Due to the parallels in the two life cycles and the close coupling between the UI and functional core of an interactive system, dual experts seem to have an implicit reassurance that they are on the right track when they undertake activities in one life cycle which has a close parallel in the other. For example, talking about the overlap of the work activities in the two life cycles one Team C member remarked "we were sure we were doing the right thing because we had something to base our work on from the previous deliverable [in the other life cycle]. We had the product concept statement to begin with right? Based on that we just iterated [and created] the product overview statement for SE. [Similarly] we used the SRS [software requirements specification] for the [UE] screen designs." Therefore there appears to be a sense of confidence

in doing activities that were related to previously completed activities in the other life cycle. However, it can be argued that this could lead to a false sense of confidence because there is no objective opinion or review from an independent role.

Team C Summary

Based on the various instruments, it appears that using a dual expert condition such as that used by Team C for interactive-software development lends itself to the possibility of a somewhat lower overall workload due to the significant overlap among the two life cycles and higher situational awareness of the development space because the same people are performing the two roles. However, there appears to be an inherent conflict of interest between the two roles, resulting in systems with minimal functionality. This conclusion is in accordance to the client's overall impression of Team C's system "First page was nice but the rest is unprofessional/rudimentary. No imagination. Not creative. Had stuff with a box around it. Very minimal."

7.3.1.4 Hypothesis H1.a summary

Hypothesis H1.a, in which we stated that teams using C condition would perform better than those using A condition, was not verified. Inherent conflict of interest and cognitive dissonance implicit in team members of C condition seems to prevent their producing a quality system and having a quality process.

7.3.2 Hypothesis H1.b

The A teams will perform better than B teams because of the amount of communication, coordination, synchronization, and dependency checking afforded by the Ripple instance to A teams and the fact that they were introduced to their counterpart teams at the start of the semester – ***partially verified***

As shown in Figure 30 and Figure 31, the A teams performed better than or equal to the mean with respect to the value index and the number of features and came in first and fourth in the overall value index ranking (Figure 30) and first and second in total feature counts (Figure 31). However, the performance was not consistent across the two teams in this condition. Whereas Team A2 performed as hypothesized and created the best software system (as rated by the clients), Team A1 was fraught with problems and required experimenter intervention. The

potential reasons for why these teams differed widely in performance, new insights into this hypothesis, and a revised hypothesis are presented in the following sections. We start with Team A2.

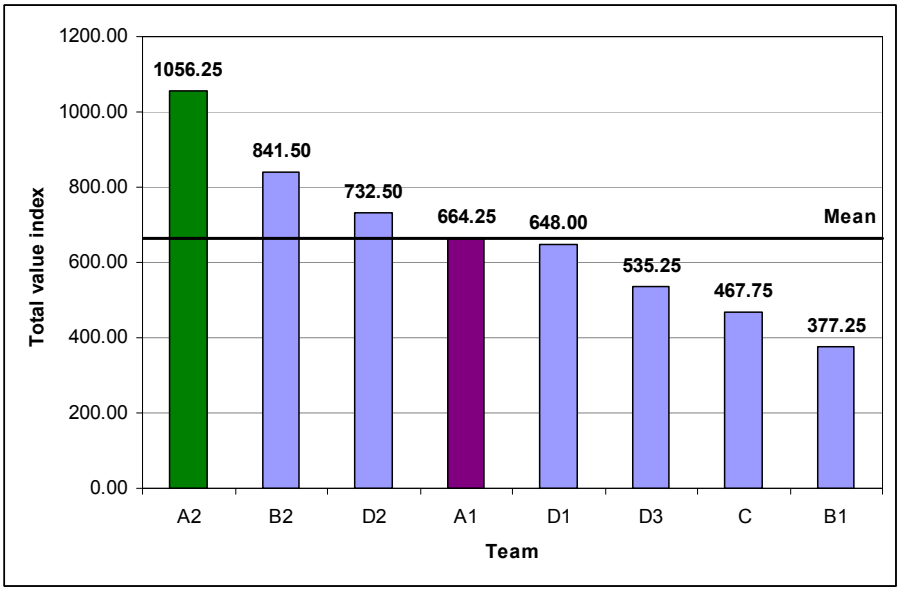


Figure 30: Total value index of all teams, showing the relative standing of the A teams

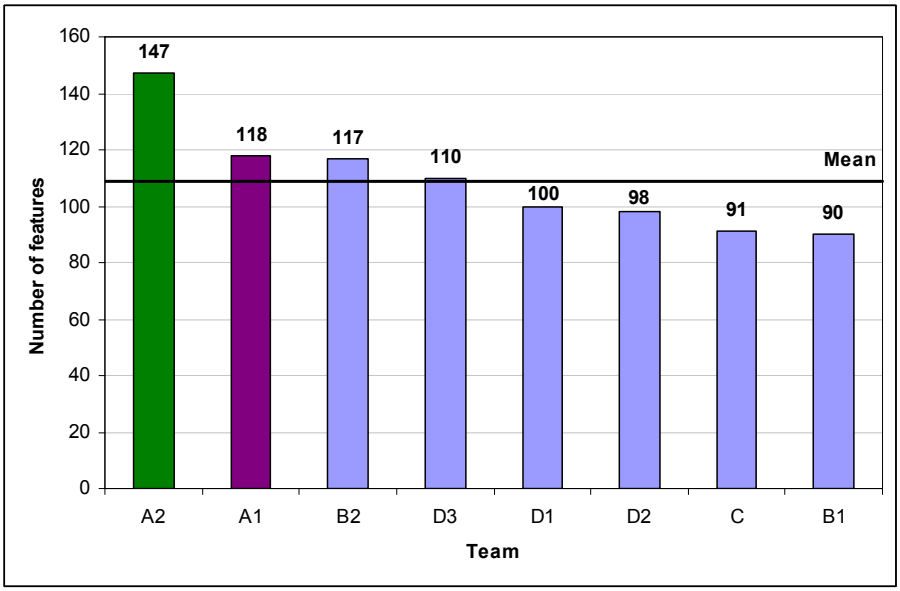


Figure 31: Total number of features per team, showing relative standing of the A teams

7.3.2.1 Team A2

Positive inter-role dynamics

Team A2 enjoyed a good quality of inter-role interaction from the very beginning. As one of Team A2's SE members commented early in the project "generally I've had good experiences working with the other team […] they have been easy to talk to and meet with when necessary and have had good ideas about the way the project should go." Such a positive inter-role dynamic was maintained throughout the project as one UE role remarked in his journal "it has been smooth sailing […] towards the end of the semester, not a whole lot [more] to say." Their combined initiative to work with the counterpart role established a good working relationship and common ground between the two roles as described by this Team A2 member: "It's amazing, for some reason both of our teams are on the exact same page for development. Every issue that has come up has resulted in no more than a five minute discussion and then a resolution that everyone is happy with. So far so good, it's been quite a relief not having to fight with each other and getting along well. After hearing about the other teams struggling to find common ground, and then to see us achieve it so easily makes me think that working in teams like this could work out, when using the right people."

Good communication between the two sub-teams

The good inter-role dynamics translated to effective communication between the two roles. As evident in this journal entry by a UE member, "[The SE role] told us they were using Pay Pal for pre-order credit card stuff. Our designs [from] the user meeting didn't reflect that decision, but it was nice to be kept up to date on their decisions, so we could change our designs," effective communication helped them manage change and minimize costly changes later on. When they encountered problems or issues they were able to quickly resolve them by scheduling a meeting as reported here by a UE member "[the] SE [role] sent us [an] email, [pointing out that they] found places where we are thinking about things differently. [We] sent him the design sketches [… and] talked about having [a] meeting to settle things. [We had a] meeting to clear up issues [and] found out they're using JSP. [We] talked about how we should make the hi-fi prototype to ease the integration process."

Effective collaboration

Each role proactively took initiative on collaborating with the counterpart role from the very beginning. As recorded by a UE role in his journal during one of the initial phases of the project "[the] SE Team did a few screen designs and brought them to the Requirements Validation meeting. This provided us with our first iteration of screen designs and a great view of what the SE Team is envisioning for the UI design." As the project progressed they maintained this level of collaboration. For example, they collaborated closely on their cost-importance deliverable where the SE team helped the UE team with cost to fix attributes in the analysis. Similarly each role made sure one of their representatives was at all the important meetings of the other role. As reported by this UE team member "[we] found out at [the] user meeting that they would like login [authentication] with VT PID and password. Since there was an SE rep at the meeting, we [were able to] just quickly ask him if that could be done [and let the client know we would incorporate that feature into the system]." In accordance with the hypothesis, their constant collaboration resulted in a successful process experience as described by this team member: "And we just kept iterating over [the design] talking with the UE team all the while saying 'hey, how is this, what can we change'? 'How can we make the next iteration better'? We were also very self-organizing in that if something came up, someone would step up and say "hey, I know how to do that, I can take care of this" and it just made for a very good overall experience."

Periodic synchronization

The two roles also conducted periodic synchronization meetings to keep their designs consistent. This team acknowledged and acted on all Ripple consistency directives (Section 5.2.2.1) that were sent to them (in contrast to Team A1). They also kept communicating about various aspects beyond the Ripple messages. As described by a Team A2 member "we also had several meetings outside the ones that just [Ripple required and would send messages saying] 'hey we all should meet up and try and condense'. And for any other meetings that say the UE team had, we would always make sure we had at least one SE representative to be the voice of the group to say 'hey, this is what is going on, this is what we think, this is where we are coming from' and then report back to the rest of the group. So […] everyone generally knew where the other team was".

Minimal problems

Each A2 sub-team had some issues with one of their team members not contributing equally to the project. In spite of this intra-role team issues, the overall collaboration between the two roles had minimal problems. Apart from the few communication problems during the first week of classes (which is probably a normal phenomenon given the volatility of students' schedules as they decide on what classes to take), there were no real problems between the two roles. When asked what problems they faced working with the counterpart role, they responded "this has been our toughest question. Basically all the ones we had were so small it is almost not worth mentioning. [The only issue we had was that of PayPal integration] that was not until a couple of weeks that we thought of but they were like 'oh! You never brought that up.' So they hadn't put that in their hi-fi prototype thing. But really it was just adding a button so it really made no difference. We also had an issue at the beginning where we [were not clear about] who was actually going to write the code and stuff like that. That was probably our biggest issue because we didn't know what our responsibilities were [and] how we meshed as a group. Then we asked you and you cleared it up for us."

Team A2 summary

Team A2 performed as hypothesized in hypothesis H1.a. The client's overall reaction to this teams system was consistent with our observations of their success in the project "they are amazing! Wow. Export to Excel feature is good thing. Had examples to show how to use the different fields on the system." Apart from the hypothesized impact of communication factors, this team demonstrated the importance of another two factors: will to work with the counterpart role and respect for that role. The gestalt of this team's success can be summarized by the remarks of on of the Team A2 members: "being on the same page from the beginning worked [well for us]. Being able to communicate since we were an A team… from the beginning of the semester helped us a lot. We really did not have that many problems. No fists were thrown, no arguments really got heated, […] The SE team would say something and we were like 'yeah. Ok'. Or we would come back with something… I mean nothing really really major happened [and] that was really nice. We contribute that, like I said, to early and frequent communication. We collaborated on the initial design, the sketches, how they pretty much stayed the same as they evolved through the semester. And we also respected each others responsibilities. The SE team respected the UE team's responsibilities of deciding what the interface should look like and what

kinds of things it should have or the way things should be laid out. We didn't tell the SE team [they] should do databases this way [or they] should run this JSP, why are you doing that, so that really helped us a lot, that kind of respect for each other."

7.3.2.2 Team A1

In many ways Team A1 was an opposite of Team A2. Whereas team A2 had an enjoyable and effective overall process and resulting product, Team A1 had a frustrating experience with a below-expected quality given their development condition. In the following sections we discuss the different factors that appear to have played a role in this team's performance.

Incompatible UI and backend components

The SE and UE sub-teams ended up having two separate visions for this project, and they never could completely reconcile their differences to create a cohesive product which was in line with the UE's design and SE's implementation. Even though the two teams were introduced at the beginning of the semester, and conducted two meeting with the clients as a joint team, their designs and product evolutions took widely divergent paths. Quoting the members from this team: "the first thing that we found was that we had different priorities from the beginning. Although we didn't have a problem deciding on doing a web interface because our visit to the greenhouse, [the client] had mentioned that we would be using just anybody's laptop who [decides] to show up that day. So that had to be a web [solution … and not] going to be any kind of a standalone application. So we did have that in common. We agreed on a name pretty early on: WebGarden is the name of our design. [But] everything else was completely different. [… We UE were] thinking about the users, the horticulture club members, what would their tasks be at the point-of-sale, we were thinking about the customers, who were the home owners, what kind of things would they like to buy? How would they like to shop for things? How easy could we make it for them? So we were all about ease-of-use because we were the usability team. Now the software engineering team had different perspectives." At this point the self-proclaimed SE lead added: "when we were talking to the clients the overall feeling that I got was that they wanted this to speed up their point-of-sale process [...] the overall main feeling that I got from them was that this system had to make their job easier and faster at the point-of-sale. So that was always at the back of my mind when I was coming up with my system design […] that it needed to be quick, it needed to be easy-to-use, very minimal, you know it just need to go 'in a snap' so

because of that a lot of our design [was about] minimalist, very efficient, very quick and easy to use. We took a lot of pointers from Google, the way that they have their web [...]". The UE role continued "and this led to a totally different idea of what shopping is! To the UE team shopping was you know, you have time to kill, you were looking for some plants, maybe you want to pick pink or purple, you are browsing around, but for the SE team it was like, a serial killer is chasing you and you had to buy a rose plant. I mean that was what was behind what they were looking at. So surprise, our designs were nothing alike." As can be observed, the UE role discussed what happened from the perspective of the entire UE sub-team whereas the SE role spoke in the first person, indicating the fact that he (one SE person), almost single handedly, steered the SE's design according to his perceptions, biases, and opinions. These differences in goals and behavior, among other things, led to almost incompatible designs from the two sub-teams, designs that could not be integrated in the end.

Disjointed SE sub-team and lack of team initiative

The A1 SE sub-team had many team problems even amongst themselves. They did not share the same vision or engage in a collective collaboration to create a quality functional backend. When one of the SE members self-proclaimed himself to be the lead, and even took control of the UI design, the other SE members did not attempt to discuss the issue but followed the lead. Lack of cohesion between the UE and SE roles prevailed throughout the project as exemplified by an SE member's statement: "The UE guys suggested we have a name for the system. [The] SE [people] are not very enthusiastic about it. But I think it's a good idea to have one." Lack of cohesion within the SE sub-team also caused issues pertaining to the quality of deliverables as reported here: "we got a poor rating on our SRS because we didn't spend enough time on the integration. The work ended up being patchy in spite of spending a lot of time on it." Divergent views of the product and process led to issues pertaining to the broader vision of the project as reported here: "my team members always think that our product will not be used by the clients. I agree that we may not be able to give 100% working features, but at least we can attempt our best. Also, they keep repeating that all documents produced so far are a waste of time and we should have instead got more time to code." The roles never seemed to work together as shown in this journal entry: "we desperately need to meet the UE guys to know where they are heading. We SE guys are making good progress and should get down to coding soon as the low level design is ready."

Uncooperative self-proclaimed SE lead

One of the fundamental reason for this teams failure to produce a quality product befitting of an A team was an uncooperative SE member who became the self-proclaimed lead of the SE team. The other two SE members in this team noticed and anticipated problems because of this but did not take any action to make their concerns heard. A journal entry in one of these two members noted "***** thinks that the UE designs are not good, so he plans to do the UE part on his own too. I think when we have been placed with a counterpart UE team, its better to discuss these issues with them first and come up with a common solution. If not, then matters can be escalated." As the project proceeded, with this kind of arbitrary action by this individual, his lack of interest in including UE inputs, his lack of respect for the UE people, and his unwillingness to communicate, matters did escalate and the communication breakdown between the two roles became more severe. After many failed attempts by the UE role to work with the SE role, and after the self-proclaimed SE lead not attending most of the joint meetings, there was one meeting where everyone finally got together. However, the SE role was still reluctant to incorporate any of the UE's designs as recorded in a journal entry of another SE member: "Looked like ***** didn't implement the suggestions from the previous meetings. I think the major problem we had is ***** is a UE person taking the SE course. He should have identified his role for this project as a Software Engineer rather than a Usability Engineer. He kept thinking from both angles which led to this asynchrony. I am also not sure how many of the suggestions (changes) he will agree to actually implement now. If he would have allowed me and ***** to touch the front-end, we could have at least caught some common ground by reusing the UE screens."

This lack of interest on the part of the self-proclaimed SE lead to work with the UE role stems from underlying biases and perceptions he had, as reported in his journal entries: "we (at least I) have not really been paying attention to the UE guys since their input does not really affect our current progress. I'm 99% sure I will just end up scrapping their UI and writing my own since I want to use AJAX and I don't think they even know what that means. AJAX is very difficult to implement if your UI is not designed for it, as all your content is dynamic. I consider myself primarily an HCI person, so I suppose I think I can do a competent design job on the UI (I like to do my own design work). [I] haven't really brought this up yet [with the UE role and] what they don't know won't hurt them."

On the UE side, there was frustration and resentment towards the SE sub-team due to their apparent apathy towards their designs as noted by a UE member in his journal after attending an SE's demo to the GTA of the SE class: "We were, or at least I was completely disappointed with the prototype developed by SE team. They seemed to have completely neglected our design and their model seemed to be a lot different from the one we proposed. The pitiful thing is that despite the fact that we shared our design, artifacts, paper prototypes, findings and feedbacks of our meetings with the clients and all other relevant information with them long [time ago], they did not seem to consider this information at all. The SE team's rationale for not adhering to our design was that they did not have time to implement our design specs. The irony is that, they implemented some of the stuff which is not a part of our design nor the client's requirements. I was really surprised with the SE team's attitude and their level of commitment to the project."

About two weeks after this demo to the SE GTA, the two roles finally met and looked at each other's designs in detail, it was evident that there was almost no overlap in the two designs. A UE member noted in his journal "there are two important issues that I discovered in the meeting. First, they had no clue of our design and client requirements, they seemed completely ignorant of our earlier feedback. It was quite obvious that they did not consider our design while they were implementing [their] model. Second they said that they were short on time and could not adhere to our design specs which seemed so ridiculous because as I mentioned earlier, we gave our specs and have been giving our feedback to them since [a] long [time]. The time and effort that we spent designing our low and high fidelity prototypes, the long discussions on design issues, all seemed to have been futile." Another UE member noted: "The SE team's version looks nothing like our version. ***** was planning to do searching by matching on a few letters the user types in. We pointed out that the user might not know the name of the plant. He said they are all horticulture majors, so they should, but that's not what the clients told us. Also, the preorder customers might just want to browse for plants by picture or type. The UE team doesn't have to implement the search, so we're only thinking about the user. The SE team is thinking about the speed of the search (which is a good thing), but not about what the users know or don't know." This lack of SE support combined with a lack of proper understanding of the client's requirements and UE guidelines resulted in a disastrous experience and resulting system.

Futility of the Ripple Instance when ignored

It should be pointed out that being an A team, the members of Team A1 got periodic messages from the Ripple instance to facilitate various forms of communication such as collaboration, synchronization, and dependency checking. However, they were either not taken seriously or, worse, they were ignored completely (by mostly the SE role). For example, one SE team member remarked "we forgot to discuss the [Ripple] project directives; strange that nobody among us pointed that out" and in another entry: "***** and ***** both failed to turn up at the UE screen walkthrough. I couldn't attend it because I had a class during that time, so I emailed them early so they could plan ahead. Both of them agreed to attend it. It's sad that neither of them remembered." This lack of consideration for Ripple messages was more pronounced in the SE sub-team than in the UE sub-team. The UE role tried to adhere to the Ripple directives as much as they could but, without reciprocation, their attempts were futile.

Lack of role and expertise distinction

As mentioned in the previous sections, the SE sub-team of Team A1 suffered from a severe problem of not distinguishing the different roles and respecting the expertise of each role. The SE role over-reached from their assigned role and decided to design the UI for the system on their own. But since the UE had no control on the code, all they could do was attempt one-sided collaboration with the SE role to get their designs adopted. One issue that compounded this problem was the fact that none of the UE members had prior web design experience. The prototype they created was not as polished in terms of look and feel as the SE's system leading the SE role to conclude the UE's prototype was not worth considering. The UE role put a tremendous amount of effort into creating the prototype that, even though looked unpolished, had a significantly better design. However, it was largely ignored. This lack of respect for the UE role and the unwillingness to work with them were the single most important reasons for this team's failure.

Team competence and experimenter intervention

So with all the above discussed problems with communication, intra-role collaboration, respect and willingness to work with counterpart role, how did Team A1 manage to create a product that is in the top half of the ranking of the eight teams? Based on evidence from the various

instruments, we believe there are two primary reasons: individual competence of the team members and experimenter intervention. We discuss these two issues here:

Team competence

The self-proclaimed SE lead, in spite of his biases and prejudices towards the UE team, is a brilliant programmer. He had substantial experience in the real-world as a professional web developer and had the ability to implement a fully functional website quickly. His other sub-team members, in spite of their lack of effort in discussing with the SE lead, were also very skilled graduate students. The SE sub-team was compartmentalized, with the so-called lead coding the UI and the others members implementing database and business logic. Therefore, when the experimenter intervened near the end of the semester (see next section), the team was able to quickly incorporate some of the core UE design ideas into their system thereby ending in a product that was much better then their originally envisioned one.

Experimenter intervention

Due to logistical issues such as the unavailability of the clients for an acceptance meeting and the fact that the teams' systems had to be evaluated for the class grades, the experimenter had to play the role of the clients' representative and conduct the customer acceptance phase with the teams. During this meeting the experimenter interacted with each system and, already being aware of the various problems with the SE sub-team's design, pointed out all the problems, noting how their lack of cooperation with the UE sub-team had not only resulted in an inferior project but also affected the UE's formative evaluation and redesign phases. At the end of the meeting, the experimenter showed how the SE design was of woefully poor quality as compared to the other teams and how they could benefit from the design of the UE sub-team. That meeting had a significant impact on the SE team and made them realize how costly their lack of collaboration with the UE role was turning out to be in terms of their grades. They immediately contacted the UE sub-team and spent about 70 person hours in the next week incorporating a prioritized list of UE recommendations.

Summary of Team A1

Team A1 was comprised of members with significant individual talent. Whereas the UE sub-team was a cohesive whole with a shared vision for the UI of the project, and zeal to collaborate with the SE sub-team, the SE sub-team was disjointed with a lack of cohesion and a self-imposed

lead who refused to acknowledge the UE role in the project. In spite of the constant sharing of work products and insights by the UE role, the SE role resisted any serious interaction with the UE role as reported in one of the SE lead's journal entries: "we spent about 90 hours combined on this project [so far] and now the UE people want to meet for 4 hours so we can do their project with them... where were they when we were coding the system? UE people are pure evil. [And] I am not meeting for 4 hours." This attitude lead to a frustrating experience and an inferior quality product. The clients remarked that this team seemed to have a lot of functionality but that most of it was not intuitive. For example, we quote a couple of remarks they made during the overall product comparison exercise: "having cart and browse from the preorder in the admin and POS systems is not necessary and confusing. It is not clear if the reports describe sold plants or plants in stock." It was not until the experimenter intervened that the SE role started collaborating with the UE role, attempting to retrofit different ideas. Since this was in the last week of the project it was not possible to make a significant impact on the overall quality.

7.3.2.3 Comparing Team A1 and Team A2

Even though these two teams were afforded the same communication by nature of the study setup, through which they were introduced with their counterpart roles at the start of the semester, and extra communication due to the Ripple instance, the two teams performed in a widely different manner. Whereas Team A1 barely collaborated with the counterpart role (for the most part fault of the SE sub-team), Team A2 was proactive in seeking help and collaborating with the counterpart role. Similarly, whereas Team A1 almost completely ignored the Ripple directives, Team A2 often went beyond the directives and met often and periodically. They even exchanged work products that were not mentioned in the Ripple directives.

Similarly while A1 had major issues with their inconsistencies with the counterpart role's design, A2 had minor issues as reported here by one of Team A2 members: "Most of our issues are really minor. Having been talking all along is making this integration process go really smoothly, I think. Everything is going well." Their willingness to work together made it possible to synchronize in such a way that the UE team was able to use the actual functional system developed by the SE role for their formative evaluation. In contrast, the A1 SE team's design barely supported the A1 UE team's benchmark tasks. The UE team ended up conducting their

formative evaluation on a prototype and their results and analysis were obviously not applicable to the SE's final system, rendering all the hard work by the UE sub-team useless.

Another important distinction between these two teams is their approach to designing the functional core. The A1 SE sub-team adopted the AJAX technology (without discussion or consultation with the counterpart UE role), which did not lend itself for modularity as reported by the SE role: "we started developing in AJAX and if any of you are familiar with AJAX, it has a lot of hooks right into the actual HTML structure of a web page, so you have to do that first. So when you have to start communicating, I mean using the java script to talk to the server, you write back into the webpage." This tight integration of the HTML structure (or the UI component) and their business logic prevented easy modification. This was in direct contrast with Team A2's approach: "the modularity of our programming really helped out a lot," where they deliberately kept their architecture modular to incorporate changes as necessary.

Another distinguishing factor between these two teams is the extent of their team cohesion. The SE sub-team in Team A1 divided the work in such a way that each member had no control or access to the other members' components. For example, one member designed and developed the entire UI, while another designed and developed the databases and business logic layers. Moreover, they rarely worked together while coding the system. Team A2's SE sub-team, on the other hand, often met together at a coffee shop while implementing the system, thereby collaborating at a much finer granularity of software modules.

7.3.2.4 Hypothesis H1.b summary

Our hypothesis that the communication afforded to A teams will facilitate their performing better than B teams was partially verified. However, we found that there are two factors that preempt the communication factor: *will to work with the counterpart role* and *respect for that role*. In spite of all the communication afforded by a particular development environment, success is actually defined by the people who have to utilize that communication.

7.3.3 Hypothesis H1.c

The B teams will perform better than D teams because B teams will have design ideas brought together from two domains, albeit late in the process and without any outside help from Ripple, whereas D teams will have a narrow user-interface-only design focus – ***not verified***

We did not find substantial evidence to support this hypothesis. It appears that our initial assumption that D teams will have a narrow user-interface-only design focus is unfounded and that the lack of a counterpart SE role actually *removes* potential implementation constraints and encourages a broad and rich interaction design by the UE roles. However, this constraint-free approach as facilitated by the D condition could have practical problems later on in the process when the prototypes developed by the D teams are passed on to software developers (not part of this study) and implementation constraints finally do become a reality. In the following sections, we discuss the different issues that were observed in each of the B and D teams and explain why we think this hypothesis was not verified. Similar to the two A teams, the two B teams also had a large difference in the overall quality, albeit for different reasons. Team B2 surprised us by creating a high quality system (second out of eight for value index as shown in Figure 32 and third out of eight for overall feature count as shown in Figure 33) in spite of the constraints placed on their development condition.

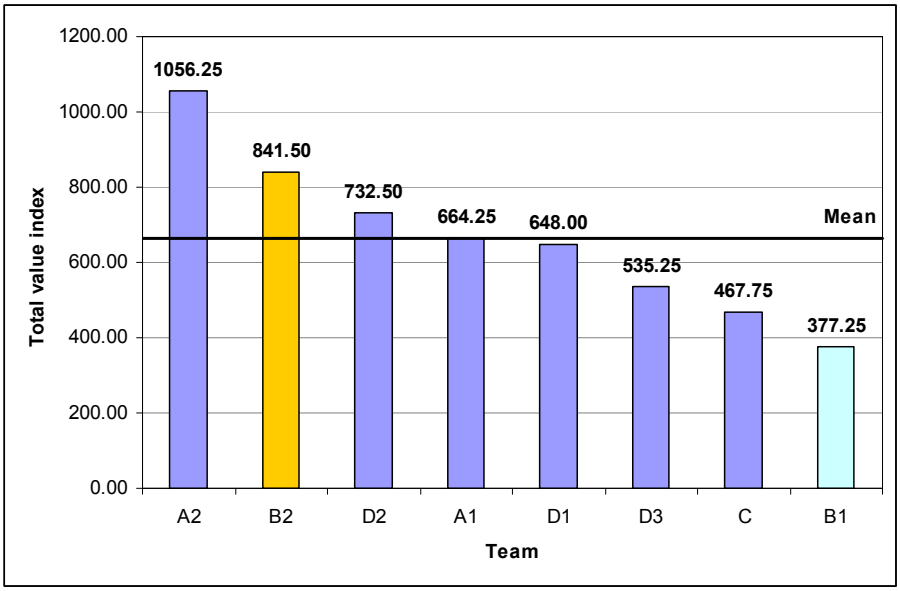


Figure 32: Total value index of all teams, showing relative standing of B teams

7.3.3.1 Team B2

Team B2 was a surprise in the study. We did not expect a B team to create a product that would end up to be a serious contender for the best project that can be deployed for the Horticulture Club's annual plant sale. However, they did, and it was a close second. Only after a long deliberation did the clients and the Horticulture Club governing body decide to pick Team A2's

system as the final winner over B2's system. This decision was before the quantitative analyses such as feature count and value index were completed and before we knew the actual rankings of the teams (Figure 32 and Figure 33). In the next section we explain their success by considering the various factors that seem to have helped in their success.

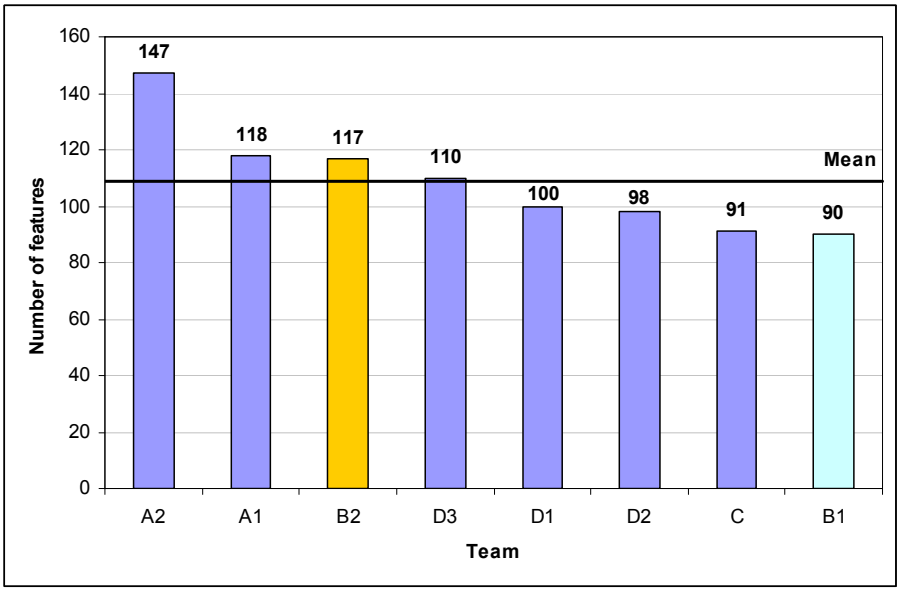


Figure 33: Total number of features per team, showing the relative standing of the B teams

Power team with highly competent members and prior shared-experience

Even though we tried to balance each team in the class as much as possible, Team B2 ended up a power team with highly competent individuals. In the demographic surveys used at the start of the semester for team formation, we asked students to rate their skills on different aspects such as previous work experience but not the specifics of that experience. Therefore we had no way of knowing, for example, if two or three people in the class worked together before this class. So unknowingly we assigned to Team B2 three students (one from SE sub-team and two from UE sub-team), who had previously worked together in a real-world software development company. This prior experience led to a level of rapport that could have helped this team achieve a high quality product. We show a part of the interview transcript where these aspects were discussed:

Experimenter: Do you think your prior experience working together helped?

SE member 1: I think it did help a little bit, because we did have some shared experience. I don't know if he [UE member 1] did, but [UE member 2] and I also did our undergrad here at VT.

SE member 2: I [also] felt comfortable right off the bat talking to [SE member 1] about whatever because I didn't really know him coming in, but I knew who he was [because I did my undergrad at VT too]

SE member 1: So a lot of us already knew one another either from undergrad here or from work, so that helped, we didn't have to establish synergy. We had some shared experiences.

UE member 1: Yeah that is true… we didn't need introductions that other teams probably did. [UE member 2] was there in my team too [who worked with me in the real world]. So we did not have any problems communicating.

Apart from their prior shared experience, this team had very competent members who adopted good software development practices to account for their lack of awareness (until late in the project) about their counterpart role's design. The SE sub-team undertook a thorough analysis initially and designed the backend with the potential for change in mind. As one SE role put it: "We knew upfront that this [project] was going to be a bit of a challenge, especially since we didn't see UE guys' design. [Therefore] we did a lot of work upfront. I don't know if you [the experimenter] remember, but our SRS was 75 pages, design document was over a 100, I think. We did a lot of work and planning up front, a lot of diagramming, just so that we would have our mind wrapped around the system from the very beginning. The diagramming, I remember doing a couple of iterations, we threw diagrams back and forth, and just ideas, workflows, whatever we were doing, that helped us a lot. All three of us on the SE side, before we met the UE team, had a pretty good idea of what [the backend] would look like down the road."

Apart from their thorough planning and good design, they were also creative when confronted with challenges and adapted to new technologies. For example, as reported by one SE member, when they faced challenges, instead of struggling with problems, they investigated the availability of better solutions elsewhere: "in-place file upload was […] more difficult than it should have been, which was a problem I think with the JSP implementation that we were using in Tomcat. And we finally ended up using a third party solution 'Jakarta Commons Framework'.

And we were fairly unfamiliar with the technologies required in order to make all this work [but adopted them anyway because they were a better solution]."

Their general competence, thorough planning, and agility in dealing with difficult challenges appear to have played a significant role in this team's performance in this study. When asked how they managed to do such a good job given all the constraints with the B condition, a UE team member, looking at other nodding UE members, responded "to compliment the SE team, I think their architecture […] They did a good job integrating our design into their architecture." A similar opinion was shared about the UE sub-team by the SE sub-team. The overall competence of these two sub-teams contributed significantly to the success of this team. One SE team member remarked: "If we were an A team and incompetent, we would have failed. We trusted [the UE's] design and [they] did a good design. We didn't have to worry about 'ok we don't really trust these UE guys,' so we will do everything ourselves. That seemed to be a problem with a lot of teams. There were good SE teams and bad UE teams and there were good UE teams and bad SE teams. I guess we sort of got lucky on that we ended up with a good UE team and a good SE team."

Inertia of inter-role interaction

Interestingly, the inter-role rapport within Team B2 did not start in such cordial terms. The first meeting between the two roles after they were introduced was a disaster. There appears to be certain inertia in getting around to working with a different team after being part of one group for two thirds of the semester. This inertia was prevalent even before the actual meeting took place as recorded by one SE member in his journal: "It seems like it's taking us forever to get a meeting set up. I suspect this is due to the fact that the groups are independent (or have been up until this point). Since the groups have strong intra-group cohesion, it may be harder to form inter-group cohesion, even though our groups do have some interrelation thanks to the connection between myself, *****, and ***** [from our prior experience working together]." To add to this inertia, there was an unexpected communication breakdown in negotiating the time and place of the first joint meeting. This breakdown resulted in only one SE and one UE member attending the meeting. In that meeting they realized that their designs were fundamentally different because the UE envisioned a standalone rich-client application for the

inventory system and point-of-sale system, whereas the SE team designed the backend components for a web-based solution.

This communication breakdown about the first meeting and discovery that their designs were widely different combined with the inertia of inter-role interaction led to some serious teething problems, with the two roles initially resenting one another. An SE member recorded his perceptions on the first meeting as: "The first meeting was incredibly disappointing. Not only did almost nobody show up, but the entire thing was a comedy of miscommunications and miscues. The fact that our teams were not synced up as to the format of the deliverable was especially vexing. I'm sure UE is unhappy that a lot of their design is targeted towards the wrong platform, and I am unhappy because we will have to adapt a UE design for a rich client to our Internet platform. Also, I was shocked at how poorly we were able to communicate. I had assumed that, given that I knew *****, we would be able to have an amicable discussion and resolve differences easily. Instead, I felt as though I was banging my head against a brick wall. Originally I was very optimistic about dealing with an apparently talented UE group, but now I feel as though having to adapt to the dictates of a separate user design group is going to be the biggest challenge in this project."

Role distinction, respect for counterpart role, and luck

So how did a joint-team that started on such disastrous note, accomplish a quality product at the end? It appears that the key reason for their success given all the constraints of a B team was their resolve to respect their counterpart roles and to keep the two roles separate. As one UE member remarked "we had good separation between the roles. [The SE members] were willing to put an effort into our usability decisions but would accept what we said in the end." Another UE member (a former VT undergraduate student now working full time in a software development company) with real-world software development experience commented: "I very early on told myself that I am not going to question them at all on SE process, which I did not. Even though I wanted to come out and say 'what is it you guys are doing?' I very early on said that I am a UE guy and [will] keep [to] my thing. I mean even ***** [who worked in the same company as a software developer before] who knew this stuff, […] was in the right frame of mind to suggest only [on functional aspects] and they were really open with the Java side of thing. They even were willing or wanted help from him." An SE team member remarked "there

was not a single meeting where we discussed about how [to design] the [user] interface to date. In our [joint] meeting[s], we always discussed about, 'ok this is it. How do we get here?' That's it. What is it, how is it, it [didn't] matter to us."

Apart from this clear separation of concerns where each role respected the other and offered to help or asked for help when necessary, without imposing or acting unilaterally, Team B2 was also a little lucky with respect to how the two sub-teams separated their concerns. As one SE member mentioned: "one thing that definitely helped was that [the UE] had broken [the system] down the same way that we did. So we didn't [have] to make a lot of changes to our underlying architecture. We tried to make it modular so that we could adapt if they did something completely bizarre, but thankfully we didn't have to test that out because they didn't." When asked what he meant by dividing it the same way, he continued: "like the preorder, point-of-sale, and then they also had the same sort of tasks in each different subdivision. Like they did not want to combine POS and preorder in any sort of weird fashion." A UE member added: "[…] I think we really ended up sharing the same elements on how we think the system should be used and I think this really took out a lot of arguments which would have happened [otherwise]."

Reuse of UE's prototype system's code in SE modules

As mentioned in the previous section, the two roles offered suggestions to one another and in one case a prototype whose code could be very easily reused in the real backend. The UE role developed a high-fidelity prototype whose look and feel and behavior was very close to the final system. They needed to build a prototype of such high fidelity (and functionality) because their design incorporated complex UI behavior such as auto-complete of form fields which could not be faked easily. Therefore during the implementation stage the SE benefited from the UE's prototype as remarked by an SE member here: "well [we] completely [took] the preorder [code and] translated [it] for the inventory." Another SE member added: "anything that we ended up using was preorder code that we adopted to [the] inventory [component]. We couldn't use most of the C# code that ***** wrote for the hi-fi prototype. Obviously the prototype was very referencable but not implementable. We could look at it and say, yeah this is how it is supposed to work but we couldn't take [actual] code from it."

Negotiation and inter-role collaboration

After the two roles resolved their initial problems with interaction inertia and started to respect the other role, their experience improved significantly. Apart for that, or rather as a result of that, they had good collaboration and negotiations with the counterpart role. As one SE member remarked: "I was really pleased with the supportiveness of the UE team during our last meeting. I was expecting them to be upset at the number of their suggested design features we had to completely ignore, but they were very understanding about it and also very nice in their praise of our efforts. I suspect a lot of this is due to the fact that the UE team is composed of people knowledgeable enough to understand the effort which went in to the creation of what we have. In an interdisciplinary team, I think it is key to foster understanding about the effort each member is putting into the project, so that then that effort can be properly appreciated." Another SE member said: "[we] reached a consensus on several aspects of the user interface design. [We also] discussed about various functionalities that could be given more importance due to time constraints. It was a very good first meeting with the UE Team. Unexpectedly everyone was willing [and] ready to discuss […] matter[s] and compromise on several issues. Everyone took equal interest and contributed well during the meeting."

Challenges arising from communication constraint

But of course not everything was roses for Team B2. Two key issues arose due to the lack of communication with the counterpart role: lack of consensus on software architecture and platform, and lack of awareness of counterpart role's interpretation of the project requirements.

The lack of consensus on the software architecture and platform led to a mismatch in the type of preorder component adopted by the two roles. One SE member recorded in his journal: "UE software implementation [of the prototype is in] C# […] for Point of Sale and Management, because UE assumed we were doing rich clients on some subsystems. They went through quite a bit of the implementation for this so some work was lost!!! [However our solution is web-based and] […] the implementation is in JavaScript so ****** is trying rigorously to implement the same aspects in his JavaScript layout (copying a lot from previous Preorder code)." Another SE member remarked: "One of the biggest problems that occurred during the building of this project was that the UE team was not aware of the platform we were going to use for implementing the system. Also they were not aware that a Web application was being developed. Hence they

coded in C# and made a standalone application which was of no use in the end. It was a waste of effort and time. Proper communication should have been made to the UE team and they should have been aware of what the final system would be like so that they could work on those lines."

Another problem Team B2 encountered because of this lack of consensus on the software platform was during the UE's formative evaluation. Quoting an SE member: "another problem was on the UE side with the formative evaluation. Since as I said before, they thought we were doing a rich client they were unable to do the formative evaluation with what we have developed because it didn't very well match what they have developed in their hi-fi prototype. They ended up having to do their formative evaluation using these prototypes and as a result it was difficult to incorporate [their] results into what we have developed because some of the issues they encountered no longer existed in the browser client, others of these issues we had already fixed and others of them were just completely unfeasible given the differences in the platform. So then the cost-importance analysis that they created ended up not really applying to our product and this caused a little bit of a disconnect as well."

The problems with lack of communication were not limited to the mismatch in platform alone. The second major issue Team B2 faced, which led to the two teams having challenges throughout the project, was lack of knowledge of how the counterpart role interpreted the requirements for the project. One SE member said in his journal: "Also I repeat that some sort of interaction should have taken place between the two teams after SRS to discuss the requirements because they were not aware of several functionalities in the system that we included [that] they didn't implement [in the prototype]." Another SE student remarked after the requirements analysis phase: "During many of [our] group meetings, we somehow felt [it] compulsory on designing the system so that it would not hamper our design when the UE team joined in. At every stage while eliciting the requirements, we discussed about what could be the implications on our design if the UE team did not agree on some of those aspects. I personally feel that if given an opportunity, our team would like to meet with the UE team [at least] once. This would assure us that the work put in by us would not be completely refuted by the counterpart team, especially predicting that the UE team will have sky high demands on the design of the system in terms of usability."

Summary of Team B2

Team B2 was composed of extremely competent members who created a quality product given the lack of communication with their counterpart role until two-thirds of the way into the semester. Initially they had problems working with the counterpart role but after the first meeting they started to respect the counterpart role and to keep the two roles separate. Two of the UE members had significant software development experience and offered advice and in come cases actual code to the SE team. The SE team worked hard from the start to create a design that would be change tolerant. Even though they faced many challenges because of their lack of communication until after the design phase, they had good negotiation and collaboration skills and managed to make a success out of a difficult project. As summed up by the client "Wow, that is pretty! Like this one. Good layout, good use of white spaces, good affordance for options." This team's performance would most certainly have been much better if they were afforded the same level of communication as provided to A teams by introducing their counterpart roles at the beginning and via the Ripple instance. One team member concluded: "So basically, communication was the root of all our problems. We didn't have any extra problems other than that I don't think. […] [Because of the lack of communication we had to] put in a lot of time [and effort to make this work]. Especially if we actually had the extra communication [like A teams] it would have been even better. If we had time to exchange requirements with each other, we could have actually fixed a lot of things that we had to spend time fixing later. In favor of [your B team] hypothesis, as we said in the symposium presentation, communication was definitely something we needed more of. I think the consensus we reached was that we wished we were an A team."

7.3.3.2 Team B1

Team B1 performed as we expected a B team would. They came last with respect to both value index and feature count metrics. This was not surprising given that they were not able to communicate with their counterpart role until two-thirds of the way into the semester and that they did not get any messages from the Ripple Instance. We discuss how we think this lack of communication and some other factors affected this team's performance.

Differences in scope and vision between the two roles

The UE sub-team in Team B1, from the start, had a broad scope for the project. Their vision for the system included, apart from the three core components of point-of-sale, inventory control, and online preordering, designs for user donations for the Horticulture Club, virtual maps showing where each plant category is located in the different greenhouses on the day of the sale, redesigning of the existing Club's website, news feeds about information that is of relevance to the Club, and for events feature on the website for the Club to post information about upcoming activities. This contrasted with a more modest scope adopted by the SE sub-team. As an SE team member summarized: "So then once we actually started [interacting with the UE team] and getting together we realized that the biggest problem that we had was, what we settled on the term, tunnel vision. The SE side just went and had their own complete view of things and we didn't really consider the UE side of things as much and the same thing on the UE side. They had their own design goals, own vision, and they didn't really consider the SE side so it was pretty much both ways."

The broad scope of UE's design further compounded an already difficult project given the constraints associated with the B condition. As remarked by an SE member: "And then being a B team, the enforced lack of communication, we felt, really hurt us. We could not come up with the necessary views and ideas that we wanted just by having… like only [less than] half of the semester to work with them. So that was really frustrating for all parts I think. Then, one of the weirdest things for us was the scope of the design, we came into our first combined meeting expecting to only having to do a point-of-sale, inventory, the standard stuff we talked about all the semester, then they told us they wanted to rewrite the entire site as well as having a news site events, and a whole host of different things."

Interestingly, the two roles did not reach a quick decision to cut the non-essential requirements but argued at length about the subtleties in the various non-essential features and why they the views of the two roles were different. A UE member noted in her journal: "***** from the SE group wanted to combine the "News" and "Events" sections into one, just for the sake of efficiency. He thought that they were one and the same. The UE group's reasoning for having them separate is that the "News" section would consist of articles written by club members or any press [coverage] that the club receives, for example, through The Collegiate Times. The

"Events" section would let the public know about upcoming activities, speakers, etc. The issue was whether "News" and "Events" are two different features. We discussed that if the club members wanted a section for articles, then a "News" section would be appropriate. If only upcoming events and other blurbs were to be used, then the two sections can be combined into one. Terminology will need to be worked out at a later date."

Interaction inertia

Similar to the effect noticed in Team B2, this team also appeared to have an initial inertia in working with the counterpart role. Even small changes lead to resistance to a certain degree as can be observed on one SE member's journal entry: "[The] UE team came up with some additions and changes to the layout and features. [These] changes are mostly minor and are in the process of being effected. News of the changes was not really alarming but we didn't like it much, especially since it was so late into the project that we were introduced to the UE team and suddenly we ended up having to make changes. Extra work for us but we should be able to accommodate most of the changes requested."

Even though this inertia against embracing and adopting the counterpart role's designs was not very severe in either of the two B teams, the experimenter still detected it. It appears that in situations where a team spends considerable amount of time and effort creating a product, they tend to be attached to it and have trouble embracing opportunities for changes in that product, even if it may be for the better. We quote here a philosophical discourse by one of the SE team members: "As we approached the meeting with the UE group, there arose questions about whether our design will be the one which is implemented or whether we will be forced to make changes after discussing with the UE team. One of my team mates was kind of reluctant to meet the UE group. Signs of conflict maybe… Although this is a more casual environment [compared to the real-world], I could easily imagine the conflicts that may be prevalent in actual work environments in the industry. Quite understandably, everyone wants their own design to be implemented. No one wants their ideas to be snubbed, especially the B-teams who have done a lot of work independently. It is quite plain to see that it is part of human nature to compete for supremacy and it will not change, so maybe we can say that there is not going to be any end to conflicts between different teams working together, more so in the actual workplace in the industry."

Territoriality issues and lack of respect for the counterpart role

Even though the two B teams were subject to similar constraints and resulting effects such as interaction inertia, they handled them in different ways. Whereas Team B2 members quickly shrugged off these problems and forged a professional partnership with their counterparts, Team B1 struggled with territoriality issues and other inter-role communication problems. For example, an SE team member unilaterally decided to make drastic changes to the UE's hi-fidelity prototype because she thought everyone in the team "could work together like one big SE-UE family and thought [she] can tell [the UE team] 'no this [design] doesn't work because I did UE [work] before for a company'." Even though the UE role agreed that at least some of these changes made the UE design better, there was a lot of resentment on the part of the UE members. As one member recorded in her journal: "when we went to look at the prototype and go over things with the SE Team, we found that the site had been completely changed!!!! ***** and the other [SE] guy had no knowledge of what happened. ***** had completely changed the layout, color, pictures, flow and everything!!! I, along with my UE team members, were very very very upset! And then to add on to that, she wasn't even at the meeting today to explain herself!!!!! [The] website […] now […] looks nothing like our prototype. Her SE members were not informed about the changes and had no knowledge of it. I'm not sure there's a resolution [to this problem] at the moment. ***** said that he would tell her to change the site back to the way we had it, with our colors, layout, etc. The change made it seem like ***** from the SE team completely disregarded the work that we had done and the implementation of what we had on the website. Everything on there had a purpose, the colors, the font sizes, the flow, etc. and I feel like she just wanted to ignore the usability/human factors side of this project. Also, it was not her job to change anything on the front end!!! As part of the SE team, it is her job to only mess with the backend!!! I'm so mad that she did that. I liked the way we had it before and it shouldn't have been changed, especially without asking anyone else or even notifying someone about it!!!"

Later on when asked about this in the group interviews, the UE roles responded: "It was the manner in which they were changed, we did not know it was changed, that like apparently no one knew about it!" with another UE member continuing "I felt that if there something that was changed it should have been more of a group discussion type."

It was not until much later in the semester that the two teams really started to collaborate seriously and work with one another towards a common goal. However by then it was too late to salvage the project.

Challenges arising from communication constraint

Similar to Team B2, Team B1 also faced numerous challenges from not knowing their counterpart role's status, which resulted in poor quality product. We quote an SE member's comments in his journal to illustrate some issues that arose because of this lack of communication: "As a developer in the work force for the past 4 years, I have always designed the backend with the eventual [UI] in mind. To not be able to talk with the UE team until 2/3 through the semester was a killer. As such, I feel our design was not as good as it could have been because there were several things that came up that we had never thought of. Also, the interface and the flow of the UE can play a fairly significant role in the design of the data structures. Our own project suffered slightly in the fact that when our UE and SE teams came together, the flow and the design they wanted forced a partial re-write of our database schema. Due to time restraints, I couldn't redesign our backend as much as I would have liked and so I was forced to do things I really didn't want to do, such as mixing JSP and Java code (mixes the display code with the design) and also forcing certain beans and actions to behave in unintentional ways to support everything. The biggest example that is still in the code is that we went from having separate beans for the different data transactions to having one giant bean that spread across everything. This was due to the overlap of functionality, or rather the lack of a clear-cut border between different functions. As such, there is more interaction and ties between objects than I would really like to see, but the time frame would not permit me the ability to redesign from scratch."

Similarly the UE role noted that there were times when having access to the counterpart role would have been helpful in making a better UI. For example while designing for security warnings and related messages on the UI, a UE role made this entry in his journal: "We are concerned about our system security, and we believe we should talk to the engineers about what options we could have for that, and how to design the system in a way to keep system security, such as giving the user proper warning. Unfortunately, we can't solve this issue until far ahead when we actually interact with the SE."

Team problems and instructor intervention

Similar to Team C, the SE sub-team of Team B1 also had issues with a certain member not performing his share of the work and with unprofessional behavior (e.g. not responding to emails in a timely manner, not returning certain work products in a timely manner for integration into project deliverables, etc.). This led to a frustrating experience for the other SE sub-team members as reported in this journal entry: "[I] Spent some time with [experimenter], [SE class GTA] and Dr. Arthur today to discuss what ***** and I have already concluded, which is that we don't feel that ***** is pulling his weight in the group. In part, this is because of his initial contribution to the SRS documentation where he was given the problem of tackling the Behavioral Description and of the 4 pages he returned, ***** incorporated very little of the information he presented. Again, when we asked him to look at the layered architecture and break it down for the LLD document, he took 3 days to give us a diagram with no accompanying text and which was in parts wrong and inaccurate. It was included in Thursday's deliverable as an example of his contribution. I am now in the position of lightening his workload or giving him his full weight and possibly sacrificing my grade. After speaking with [experimenter] and Dr. Arthur, I was told to give him his full share of the workload and to address issues as they rise.

I'm getting frustrated. Our project has almost 50 JSP files and about a dozen java files and almost all of them were done by me. ***** has taking over a week to get the simple stuff done so now I'm backlogged doing his stuff when I was getting my stuff done on time. So now ***** and I's [sic] grades will suffer because of this, even though I've done everything in my power to get him to carry his weight. And now the [censored word] server hasn't reloaded any java I've uploaded in the last 2.5 hours. So I can't even attempt to get the stuff done that ***** was eventually going to have to get to. This is a facet of college life that I don't miss. It doesn't really help your study at all, [experimenter], but it makes me feel better to vent."

Team B1 summary

Team B1 was comprised of some competent members who had real-world software development experience. However, they performed poorly in this study because of lack of communication afforded by the B condition. They also had problems with each role having a different vision and scope for the project and other territoriality issues. Further, the SE team had a problem member

who did not contribute as much as the rest causing unnecessary pressure in a project that was already strained because of missing communication and unrealistic scope.

7.3.3.3 Comparing Team B1 and Team B2

Both these teams had formidable challenges due to lack of communication between the SE and UE roles for two-thirds of the project duration. Similarly after introduction to their counterpart role, they both had initial problems with embracing the idea of a joint-team with SE and UE roles. However, Team B2 quickly (by their second meeting) succeeded in overcoming those initial problems and started to collaborate. Team B1 on the other had territoriality issues and could not form a cohesive joint-team until much later. Also, Team B2 negotiated the scope of the project and prioritized how they would use their resources to complete the project. Team B1, however, could not narrow their scope quickly enough to have at least a reasonable system for the client.

7.3.3.4 Discussion of B condition

The lack of communication like that demonstrated in B teams seems to have a greater impact on the SE role than the UE role. This is because they not only have to design a functional core without any knowledge of how the UI would look or behave, but also have to accommodate UI designs that were not communicated to them until after the SE design phase. On the other hand, most of the challenges for the UE side appear to be related to negotiating with the SE role about features that they deem important for better usability of the UI, and incorporating their formative evaluation findings to the functional system.

Also, as evident in the case of Team B2, communication between the SE and UE roles is an important requirement to have in an interactive-software development project, and it would have improved the quality of their system drastically. However, the respect for the counterpart role, willingness to work with them, and ability to negotiate and collaborate without ownership issues and territoriality, are *essential* to the success of a project.

7.3.3.5 D teams

The performance of the D teams was another surprise in this study. We hypothesized that because of the lack of SE inputs these teams would perform the worst. However, this lack of SE input actually led to a constraint-free environment in which they were able to be creative and

create prototypes with broad designs and good aesthetics. Even though these teams only created high-fidelity prototypes, with a lot of the envisioned functionality stubbed (which was not counted in this analysis), they still managed to reach near the mean for both feature count and value index (Figure 34 and Figure 35).

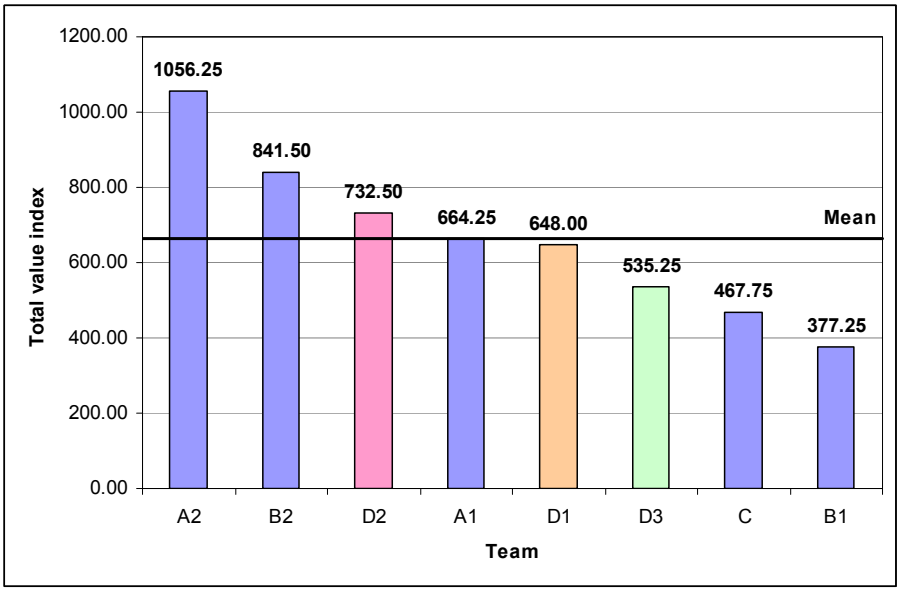


Figure 34: Total value index of all teams, showing relative standing of the D teams

Apart from a few comments during the high-fidelity prototyping stage of the UE process where some members in D teams felt it would have been nice to have an SE role to help code the prototype, they did not miss having an SE counterpart role. As one member from Team D2 remarked: "So far, I have not missed having an SE team working with us. I think we have a lot of freedom in our project decisions, and I like that. We can focus our design on the usability goals." It appears that the only limiting factor for these teams was the prototyping platforms they choose and their programming abilities on those platforms. In other words, these teams were free from the constraints imposed by a counterpart SE role, the overhead associated with negotiating with them, and other project management overheads such as scheduling meetings, but they had to do their own programming (for the prototypes). As commented by a member in Team D2: "I think in the beginning all [UE] groups [in all conditions] were in the same page [to create usable products] but as things went on they would have to focus on the merging [their designs with SE] thing [… while] when implementing [our prototypes] […] we were always trying to keep in mind our usability goals. Like checkout. We wanted it to be fast. So not only providing the

functionality [in the prototype] but we always tried to remind ourselves what […][the clients] need."

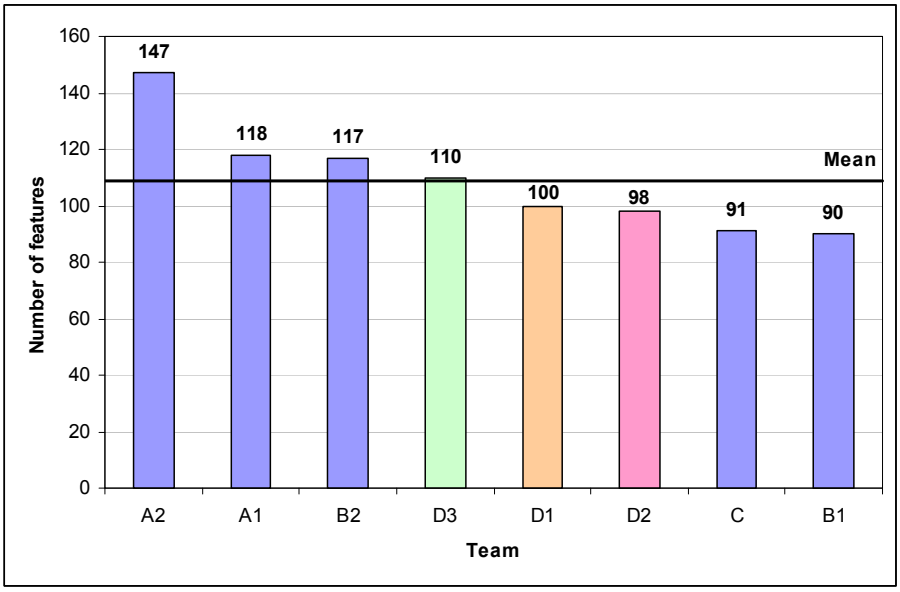


Figure 35: Total number of features per team, showing the relative standing of all D teams

This factor of not thinking about the SE implications resulting in broad and rich UI designs was also detected in the case of Team B1, only in their case it acted as a detriment to the project because of their later requirement to communicate and develop a fully functional system. An excerpt from a conversation discussing this effect in Team B1 is provided here:

UE member 1: We kept thinking about the users and what the cool features they would like to have. Like we wanted to have that map where they can say what plants are they in this building, we had like the news feed, etc. If we thought about it from a different perspective [which includes SE implications] we probably wouldn't have done that.

UE member 2: For myself, I did not think about [SE implications]. I just cared about the UI and […] the usability issues […]

UE member 3: I agree with [UE member 2], I really did not think about the SEs. I really did not know what the SEs can and can't do, what can be done and what is hard to do. So I don't know stuff about SE so I did not think that would probably take a long time.

UE member 4: For example we wanted to have something like a virtual tour. We asked ourselves who would do it? But then we felt we should not restrict ourselves

Another aspect of D teams was that they seem to enjoy the process more than the joint-teams, probably because of their creative freedom and lack of other project overheads. When asked what condition they would prefer if they were to participate in a study like this again, most of the D team members answered they would pick the same D condition again. When probed why, they responded that the D condition afforded a level of pure perspective on designing a UI, and that looking at all the challenges faced by the A, B, and C teams during the semester, they would rather have a constraint-free and trouble-free condition like D.

7.3.3.6 D teams summary

The inherent lack of SE constraints in the D team seems to afford a level of freedom for the UE role that resulted in creative and broad designs. Two of the most loved (by the clients) systems out of all eight were designed by D teams. The clients liked Team D2's design the best among all eight systems and remarked "this is the coolest design. They had a very broad design, with a lot of extra features such as zone map, etc. Their 'add to cart' is the best. The information is very detailed. Love it. This is like the real thing. Wow. Unbelievable." The clients also liked Team D1's prototype and made many positive comments on the aesthetics of the design and said "looked like they listened to us and have a broad coverage of design, good layout, with professional look and feel. Wonderful!"

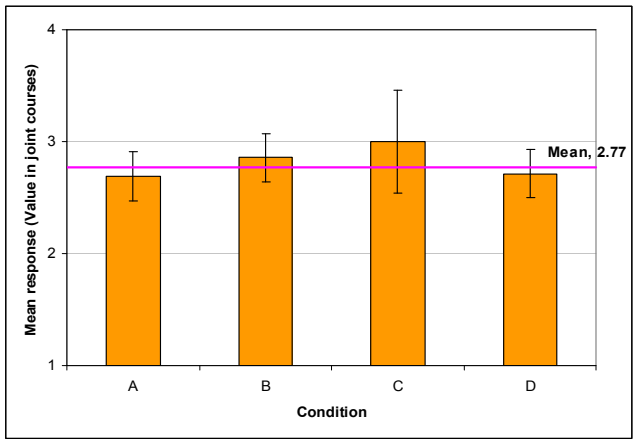
7.3.3.7 Discussion on the D condition

Based on the client reactions to D team prototypes, experimenter observations, and the fact that these teams created rich and broad designs seems to indicate that, in situations where there are fewer project constraints and in designing exploratory or emergent systems, D teams provide a good vision of what could be a good design. For example, if a company is attempting to create a competitor to an existing market leader, using a constraint-free UE-only approach might set the boundaries for what the product's design should aim to achieve. In a way, D team designs seem to have the potential to set the stage for what a successful product should be, and push the boundaries of what is possible by the SE role. Using the same argument, it appears that this type of development condition does not lend well to development of interactive systems which are not ground-breaking and with limited project resources.

7.3.4 Hypothesis H2

The students in this class will have a better learning experience as pertaining to learning the intricacies of interactive-software development than compared to traditional SE-only or UE-only curricula. – ***verified***

For this hypothesis we used the survey instrument to investigate whether students felt having a joint-class was more valuable than as compared to the traditional SE- or UE-only classes. We asked students in both classes to gauge their perceptions on this matter on a standardized questionnaire. We present the results from these questions here. The first question asked the students, based on their experience having a joint SE and UE class like the one in this study, to rate if this experience was more valuable, same as, or less valuable than an independent SE or UE class. We used the numerical values 3 for more valuable, 2 for neutral, and 1 for less valuable. A one way ANOVA was performed on student response for value of joint course by development condition and the results are shown in Figure 36. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	2.69	0.22
B	14	2.86	0.21
C	3	3.00	0.46
D	14	2.71	0.21

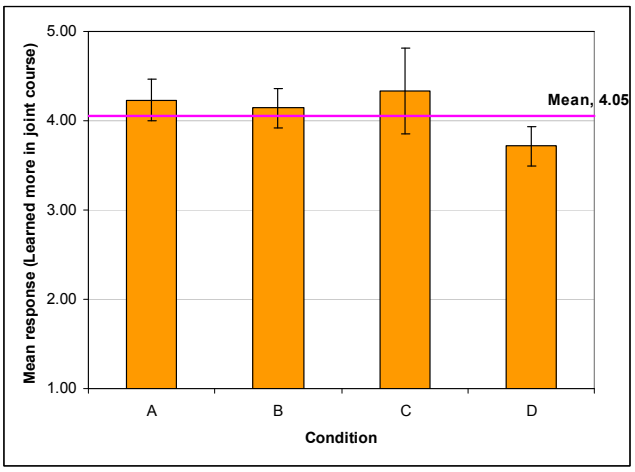
Mean of response = 2.77

Question options:
More valuable = 3
Neutral = 2
Less valuable = 1

Figure 36: Mean response (plus SE) for student perception of value in joint SE-UE curricula

The mean response across all students was 2.77 (out of 3), indicating that the students perceive joint courses more valuable. Even though the mean for condition D is lower than other conditions, the difference was not statistically significant. This lower mean for the D condition could be explained by the fact that the student in this condition did not have the same first-hand experience of working with the counterpart role and therefore the value for them was mostly observational.

In another question we asked the students to rate their agreement to the statement: "Overall, I learned more than I would in this class because of this "connected" offering of SE and UE classes" using options strongly disagree, disagree, neutral, agree, and strongly agree. We used numerical values of 1, 2, 3, 4, and 5 respectively for each of these options. A one way ANOVA was performed on student response for learning in joint courses by development condition and the results are shown in Figure 37. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	4.23	0.23
B	14	4.14	0.22
C	3	4.33	0.48
D	14	3.71	0.22

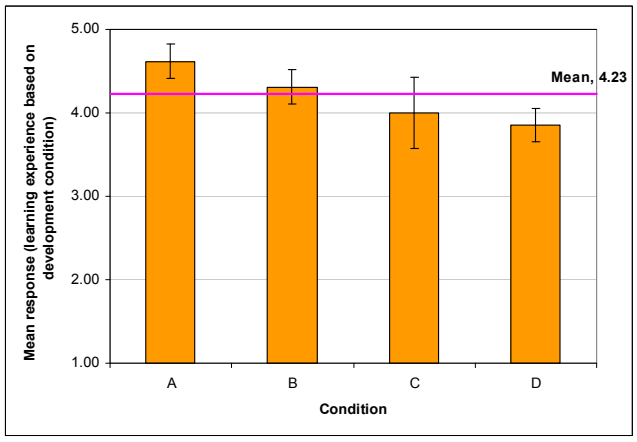
Mean of response = 4.05

Question options:
Strongly agree = 5
Agree = 4
Neutral = 3
Disagree = 2
Strongly disagree = 1

Figure 37: Mean response (plus SE) for student perception of learning in joint SE-UE courses

The mean response for this question across all students was 4.05 (out of 5) indicating that the students perceive they learn more in joint SE-UE classes than in individual ones. Similar to the previous question on value of joint classes, the mean response of D students to this question was lower than those in joint conditions. Once again this could be explained by the fact that these students did not have the first-hand experience in learning about the intricacies of interactive-software development, rather they learned about it through observation and end-of-semester symposium.

This difference in learning experience for different teams was also observed in another question (rating scale inverted here for consistency with previous questions): "Based on the type of your team (A, B, C, D) how would you rate your learning experience in this class (as opposed to what you perceive members from other teams learned from their project)? (circle a number) **Best** 5 4 3 2 1 **Worst**." A one way ANOVA was performed on student response for learning based on development condition by development condition and the results are shown in Figure 38.



Level	Number	Mean	Std Error
A	13	4.61	0.21
B	13	4.31	0.21
C	3	4.00	0.43
D	14	3.86	0.20

Mean of response = 1.77

Alpha = 0.1, F Ratio = 2.48, Problem>F = 0.0753

Level			Mean
D	A		4.61
C	A	B	4.31
B	A	B	4.00
A		B	3.86

Levels not connected by same letter are significantly different (by Tukey-Kramer HSD)

Question options: Best→ 5 4 3 2 1 ← Worst

Figure 38: Mean response (plus SE) for learning experience based on development condition

An all-pairs comparison of means using the Tukey-Kramer HSD was performed across conditions. The analysis showed that there was significant difference between conditions A and D, with students in the A condition rating their learning experience significantly higher than that of the students in the D condition. Once again, this can be explained by the fact that the D teams did not have first-hand experience with the issues involved working with a counterpart role. However, the mean response for D team is 3.86 which is still above the normal learning experience, if we assume a response of 3 to indicate the learning experience for a traditional UE-only class.

7.3.4.1 Hypothesis H2 summary

As predicted, students taking joint offering of SE and UE appear to have a better learning experience than traditional SE-only or UE-only classes. This was observed in UE-only teams as well, indicating that even if students are not actively participating in a project with a counterpart role, just being in a learning environment where different development conditions are adopted by different teams and experiences shared at the end has a positive impact on the students' learning.

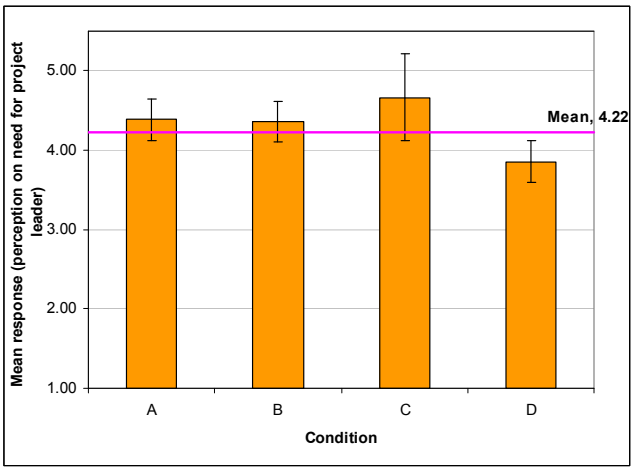
7.4 Exploratory Aspects and Potential Quality Factors

In this section we discuss factors beyond those discovered as part of our investigation into the hypotheses via the various instruments during the course of this exploratory study.

7.4.1 Need for a project leader

One factor that we found could impact the quality of an interactive-software development effort is the presence of a project leader. As one Team B2 member remarked that it would be useful to have "a project manager, someone more business related [who] probably checks on time [schedules and] […] who can keep more the customers goals in mind as much as the software engineering or usability engineering. One thing […] [that] will benefit [a project] is change control, [because] a lot of time [one] […] don't really understand what changes [one] […] need to negotiate on, what features [one][…] need to actually go and implement, so having a third party that kind of overlooks [those aspects] […] and lays down the communication between the two teams to decide what sort of changes need to be put into the system will […] be quite beneficial." This factor was also identified by Team A1, when they said the presence of a project leader or an active moderator would have corrected their problems in time. When asked in what way a project leader would have helped where the Ripple instance messages have failed, the said a project leader would be "active" as opposed to having "passive" messages from a Ripple instance. One Team A1 member said: "I think a project manager would have helped. You know that we were supposed to be sharing the responsibilities [and if only] […] somebody had said 'I am going to keep you people on task you are going to do this, you are going to do that' it would have worked much better [in correcting our problems in time]."

This desire for a project leader was also observed in a question in one of the surveys. When asked to rate their agreement to this statement: "Having a project leader to oversee the two roles and enforcing the design suggestions of each role would be" on a rating scale of very useful, useful, will probably not make any difference, harmful, and very harmful, using the numerical values of 5, 4, 3, 2, and 1 respectively. A one way ANOVA was performed on student response for need for project leader by development condition and the results are shown in Figure 39. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	4.38	0.27
B	14	4.36	0.26
C	3	4.67	0.55
D	14	3.86	0.26

Mean of Response: 4.22

Question options:
Very useful = 5
Useful = 4
Will not make any difference = 3
Harmful = 2
Very harmful = 1

Figure 39: Mean response (plus SE) for student perception of need for a project leader

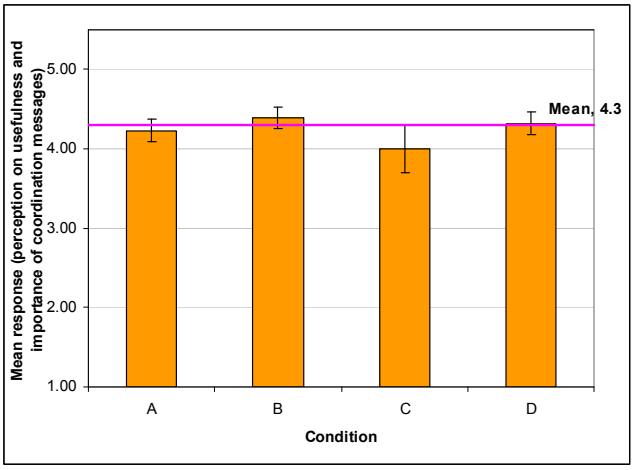
The mean of response across all students was 4.22 (out of 5) indicating a preference to have a project manager oversee the two activities of the two roles and to make sure that the recommendations of each role are enforced. The mean response from students in the D condition is less than the overall response but the difference is not significant. This lower preference by D students can be explained by the fact that these students did not experience problems with negotiation and other inter-role conflicts that were prevalent in other joint teams.

7.4.2 Usefulness of a Ripple-like frameworks

One of the overarching objectives of this study was to investigate the effectiveness of the Ripple instance. However, because of various constraints imposed by the classroom setting, i.e. being able to have only two teams use Ripple, we were not able to prove the effectiveness of Ripple with team-level data. Therefore, we resorted to gathering perceptions of students in this class, who were able to observe the various interactive-software development conditions first hand, to assess the effectiveness of Ripple.

We covered three main communication aspects Ripple provides: coordination, constraint and dependency checking, and synchronization. To assess the coordination value of Ripple communication, we asked the students to rate their agreement to the statement "getting messages informing you about your counterpart team's activities and suggesting you have a representative participate is important and useful" on a scale including strongly disagree, disagree, neutral, agree and strongly agree, and using numeric values of 1, 2, 3, 4, and 5 respectively for each of

these options. A one way ANOVA was performed on student response on usefulness and importance of coordination messages by development condition and the results are shown in Figure 40. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	4.23	0.14
B	14	4.39	0.14
C	3	4.00	0.30
D	14	4.32	0.14

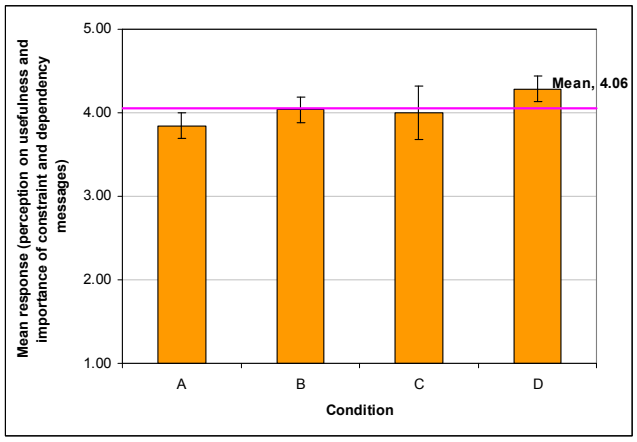
Mean of response: 4.3

Question options:
Strongly agree = 5
Agree = 4
Neutral = 3
Disagree = 2
Strongly disagree = 1

Figure 40: Mean response (plus SE) indicating student perception of usefulness and importance of coordination messages

The mean of response across all students in the class was 4.3 (out of 5) indicating that the students perceive coordination messages, like ones sent by Ripple instance, as useful and important.

Similarly, to assess the constraint and dependency checking value of Ripple communication, we asked the students to rate their agreement to the statement "getting periodic messages with specific directives on consistency of design artifacts between the two roles is important (e.g. Message saying 'make sure all SE use cases are supported by the UE usage scenarios')" on a scale including strongly disagree, disagree, neutral, agree and strongly agree, and using numeric values of 1, 2, 3, 4, and 5 respectively for each of these options. A one way ANOVA was performed on student response on usefulness and importance of constraint and dependency messages by development condition and the results are shown in Figure 41. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	3.85	0.16
B	14	4.04	0.15
C	3	4.00	0.33
D	14	4.29	0.15

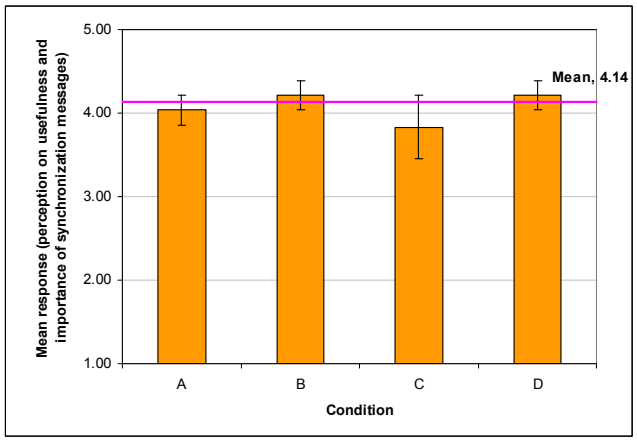
Mean of response = 4.06

Question options:
Strongly agree = 5
Agree = 4
Neutral = 3
Disagree = 2
Strongly disagree = 1

Figure 41: Mean response (plus SE) for student perception of usefulness and importance of constraint and dependency messages

The mean of response across all students in the class was 4.06 (out of 5) indicating that the students perceive messages informing constraint and dependency checking, like ones sent by Ripple instance, as useful and important.

Also, to assess the synchronizing value of Ripple communication, we asked the students to rate their agreement to the statement "Getting periodic messages with specific directives on what work products need to be ready for the other role is important (e.g. Message saying 'make sure you have your UE screen designs ready because the SE role is going into implementation next week')" on a scale including strongly disagree, disagree, neutral, agree and strongly agree, and using numeric values of 1, 2, 3, 4, and 5 respectively for each of these options. A one way ANOVA was performed on student response on usefulness and importance of synchronization messages by development condition and the results are shown in Figure 42. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	4.04	0.18
B	14	4.21	0.17
C	3	3.83	0.38
D	14	4.21	0.17

Mean of response = 4.14

Question options:
Strongly agree = 5
Agree = 4
Neutral = 3
Disagree = 2
Strongly disagree = 1

Figure 42: Mean response (plus SE) indicating student perception of usefulness and importance of synchronization messages

The mean of response across all students in the class was 4.14 (out of 5) indicating that the students perceive messages informing synchronization, like ones sent by Ripple instance, are useful and important.

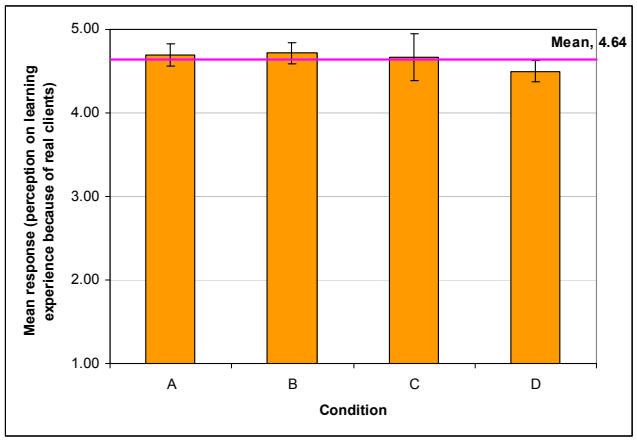
7.4.3 Pedagogical value of student personal journals

During the course of this study, we had a relatively fine-grained understanding of each team's activities and problems, how well they were doing, what problems they were facing, who among the team members were not performing their share of the work, and who among the teams were causing discord in the team. As an instructor or a teaching assistant of a course with a team project we never before had this level of micro-understanding of each team and its member-dynamics. The key reason we now gained this level of insight was the use of shared online personal journals that each student was required to maintain as part of the study. Through these journals, the experimenter was able to observe the progress of each team and identify problem teams. Based on this information we were able to intervene into four teams and correct team problems before too much damage was done. The nature of entries in these journals ranged from insights into different aspects of the project, to experiences in real-life related to the content of the course, therapeutic discourses on individual frustrations, comments about problem team members, and even philosophical discussions on how SE and UE roles should or should not interact. It was surprising how normal some of the problem teams appeared to be on the outside,

and when they were interacting with each other in classroom activities. But deep resentment and grave problems were just a surface depth away. We believe this could be used as an effective tool in a classroom setting to monitor and, if necessary, intervene into a team to correct problems. However, the use of journals was not very well received by the students themselves because of the extra overhead involved in maintaining them.

7.4.4 Use of real clients for group projects

Another pedagogical factor that seemed to motivate and interest the students in the class project was the use of real clients. We quote members from Team A2: "it was really nice to work on a meaningful project and actually have a client to go and talk to. Instead of having like this… imaginary [or textbook description of a] person who we were supposed to pretend is somebody who we are designing this product for. Being able to go there and get real feedback from was really nice. [Member 2] It gave us a motivation and a reason to actually care about it. [Member 3] Yeah, that whole appreciation, that praise, from the client at the end of the meeting was always a nice boost to our confidence." We found that this preference to work on a real project with a real client to be prevalent in all conditions. When we asked them to rate their agreement to the statement "working with real clients for the class project made the learning experience better" on a scale of strongly disagree, disagree, neutral, agree, and strongly agree, with numerical values 1, 2, 3, 4, and 5 respectively. A one way ANOVA was performed on student response on better learning experience because of real clients by development condition and the results are shown in Figure 43. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	4.69	0.14
B	14	4.71	0.13
C	3	4.67	0.29
D	14	4.50	0.13

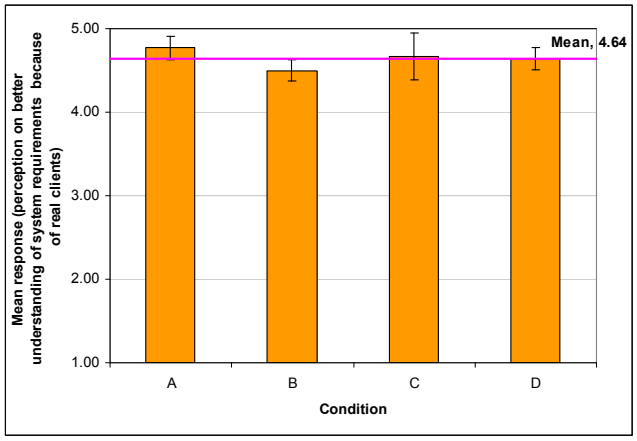
Mean of response = 4.64

Question options:
Strongly agree = 5
Agree = 4
Neutral = 3
Disagree = 2
Strongly disagree = 1

Figure 43: Mean response (plus SE) for student perception of better learning experience because of real clients

The mean of response across all students in the class was 4.64 (out of 5) indicating that the students perceive having a real client for the group projects to enhance their learning experience.

Similarly when we asked them to rate agreement to the statement "having real clients provided a more realistic understanding of the requirements for the system" on the same scale as above. A one way ANOVA was performed on student response on more realistic understanding of system requirements because of real clients by development condition and the results are shown in Figure 44. The differences in conditions were not statistically significant.



Level	Number	Mean	Std Error
A	13	4.77	0.14
B	14	4.50	0.13
C	3	4.67	0.28
D	14	4.64	0.13

Mean of response = 4.64

Question options:
Strongly agree = 5
Agree = 4
Neutral = 3
Disagree = 2
Strongly disagree = 1

Figure 44: Mean response (plus SE) of student perception of more realistic understanding of system requirements because of real clients

The mean of response across all students in the class was 4.64 (out of 5) indicating that the students perceive having a real client for the group projects provides a more realistic understanding of the requirements of the system.

7.4.5 Scheduling overhead in graduate programs

One interesting observation we made during the course of this study was regarding the difficulties graduate students face with respect to scheduling group project meeting times. Given the conflicting schedules of graduate students taking different classes in a university, almost all team faced numerous scheduling challenges. Using the group email instrument, we computed the percentage of communication units spent on scheduling during the entire duration of the project as shown in Figure 45. We found that on average team spent about 17% of their overall communication on scheduling alone.

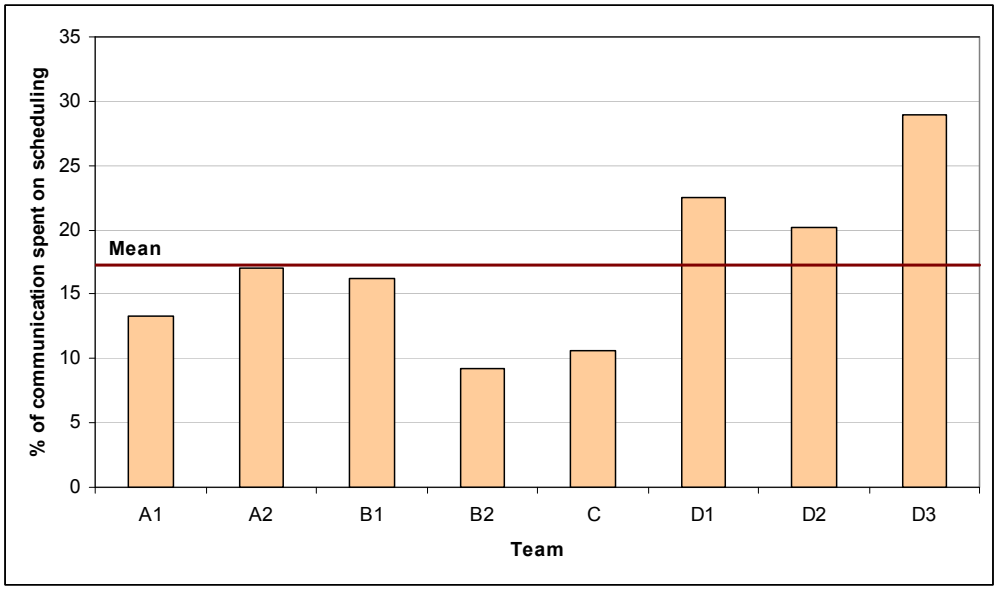


Figure 45: Percentage of communication units spent on scheduling tasks per team

Given that many graduate classes entail some level of group work, each student could potentially be spending about 17% of their time on scheduling per class. We believe this to be a high overhead that could be avoided by making structural changes to the way graduate classes are scheduled. Based on this observation, it appears that leaving a particular time of a week for group project work only, and not scheduling any graduate classes during that period could effectively solve this problem of unnecessary overhead.

7.5 Summary of Analyses and Results

In this chapter we discussed the different instruments we used in our exploratory study, the challenges associated with each of those instruments, and the need to use combination of these instruments for an exploratory study like this. We discussed each of our hypotheses and associated data. Our hypothesis 1.a that C teams will perform the best because of their implicitly high amount of communication was not verified. It appears that C teams suffer from conflict of interest and cognitive dissonance because of the need to play dual-roles in a project. Our hypothesis 1.b that A teams will perform better than B teams because of better communication in A teams was partially verified. It appears that respect for counterpart role and the willingness to work with them preempts the need for communication in an interactive-software development environment. Our hypothesis 1.c that B team will perform better than D teams because of the contributions of an SE role in B condition was not verified. It appears that the lack of SE participation actually helps D teams create broader and richer designs for the UI. However, the practicality of such an approach to interactive-software development effort could not be investigated in this study. Our hypothesis 2 that students taking cross-pollinated SE and UE courses will have a better learning experience compared to SE-only and UE-only courses was verified. Students did report that they had a better insight into the intricacies of interactive-software development efforts.

8 Conclusions

8.1 Introduction

In this chapter we summarize the key findings from the exploratory study described in Chapter 7. These findings are derived from experimenter observations and a variety of instruments discussed in Section 7.2. Given that the study was exploratory in nature and that there were at most three teams per condition, these findings were typically not statistically significant. However, almost all of these findings are supported by qualitative data that was derived from multiple instruments in the study.

8.2 Importance of Communication and Usefulness of Ripple Framework

As demonstrated by the most successful team (i.e. one with best overall value index, most number of features, and chosen by clients for deployment) in the study (Team A2), a Ripple-like framework appears to foster critical communication required for the success of an interactive-software development effort. Most students, based on their experience in this study, opined that a framework like Ripple is useful. This opinion was especially strong in students from B teams where communication was constrained until the design phase was completed. Also, the experimenter observed various instances during the study when Ripple-like communication messages played a key part in bringing the SE and UE roles together and where the lack of such messages created costly problems. As a counter example, B teams demonstrated that a lack of early communication, which Ripple fosters, between the SE and UE roles creates serious problems in the context of interactive-software development.

8.3 Inherent Conflict of Interest in Dual Experts

Interactive-software development endeavors where the same developers perform both SE and UE activities (C condition in the study) appear to suffer from an inherent conflict of interest with respect to doing justice to different aspects of UI design for a system. Because dual-role personnel are responsible for both UI design and UI software implementation, there is a strong risk of their limiting the UI design to features that are easy to implement and not necessarily those that make better interaction design. It appears that having individual roles for the SE and

UE life cycles mitigates this problem by providing a system of checks and balances within which UE roles champion the need for usability of features without a concern about ease of implementation.

In contrast, at other extreme of development conditions are the UE-only (D) teams where the developers focus on UI design alone. The lack of SE constraints during the design process in these teams seems to afford broad and rich UI designs which are only limited by the prototyping platform used and the prototyping skills of the UE role. Also, there was less communication overhead involved in these teams as they did not have the need to communicate with an SE role. This lower workload combined with the creative freedom inherent to this condition led to students enjoying the process more than other conditions. However, developing interactive software using this condition runs the risk of major surprises when the UI prototypes are handed over to the SE role for implementation due to platform and architectural constraints. Therefore it appears that developing interactive software using the D condition is more suitable for innovative products rather than traditional systems.

8.4 Factors that Preempt Communication

One of the key lessons learned in the study was how absolutely important communication among the SE and UE roles was for the success of an interactive-software development project. However, as Team B2 demonstrated, high-quality personnel, competence, motivation, and initiation can often overcome any challenge, even a lack of communication. Even though lack of communication in a project can be surmounted by a team with such exceptional abilities, the study indicates that the resulting quality of the product and process could be even higher for such teams when there is early and frequent communication.

In the study we also discovered that two factors can preempt the importance of communication in a negative way. One of these factors is lack of respect for the counterpart role that should come from an understanding about the effort each member is putting into the project. Teams that respected the abilities and skills of the counterpart role and deferred the design decisions of the corresponding life cycle to that role created better products and had better process experiences than those without. The second such factor is a lack of willingness to work with the counterpart

role. As demonstrated by Team A1, having ability to communicate is not useful if one role is not willing to work with the other.

8.5 Inertia of Interaction Between Roles That Come Together Late in the Project

As demonstrated in the B condition, when groups of developers begin a project by working in isolation from their counterpart roles, the roles experience difficulties in working with each other when brought together later in the project. Because of the inertia of their initially interacting with only their own role, team members are likely to resist sharing or relinquishing control of their work when they finally do have to collaborate with a counterpart team. The lack of a shared history prior to integration leads to territoriality and resentment of the other team. The longer the sub-teams work independently the more ownership they attach to their designs, causing them to be even less willing to change or compromise, even if those changes are for the better. Also, working in isolation with only one perspective (UE or SE) for an extended period of time runs the risk of developers having a tunnel vision of the design, often missing out on a broader perspective. Therefore, it appears that ensuring periodic communication between the two life cycles from the beginning is essential to foster overall team cohesion in interactive-software development endeavors. Such team cohesion allows the two roles to establish, agree upon, and understand the UI design implications on the software architecture and implementation platform and vice versa. Having early and frequent communication between the two roles also, of course, facilitates agreement on the overall system requirements and scope.

8.6 Need for Unbiased Project Leader

One of the recurring issues observed in all joint teams in the study was the need for negotiation between the SE and UE roles about scope and feasibility of different features. Given the tight coupling of UI and backend in an interactive-software system, the need for such negotiation and feasibility analysis is inevitable. However, because each role invests significant amount of time and effort into developing their own component (UI or backend), there appears to be a certain level of territoriality and inability to compromise when changes have to be made to a role's design. Having an unbiased project manager with good leadership skills and a holistic view of the project can help resolve the SE-UE issues quickly. Most students in the study opined that

having such a project leader would have helped them in resolving issues that consumed a significant amount of time in argument.

8.7 Importance of Cross-Pollinated SE-UE Courses

Based on experimenter observations and student perceptions in the study, joint SE-UE courses are more valuable and facilitate a better learning experience about the ways these roles must work together in interactive-software development. Simulating different conditions for developing interactive software seems to illustrate the tradeoffs and challenges associated with each condition, and semester-long projects seem to provide firsthand knowledge about the challenges and advantages of each condition.

References

Allgood, C., The Claims Library Capability Maturity Model: Evaluating a Claims Library, Masters Thesis, *Department of Computer Science*, Virginia Polytechnic Institute and State University (Virgina Tech), Blacksburg, (2004).

Alsumait, A., Seffah, A. and Radhakrishnan, T., Use case maps: A roadmap for usability and software integrated specification. *Proc. IFIP World Computer Conference*, (2002), 119-131.

Ambler, S.W., What is(n't) agile modeling, (2002), http://www.agilemodeling.com/essays/whatIsAM.htm, Last accessed Sep 10, 2007.

Ambler, S.W., What is agile modeling (AM)?, (2004), http://www.agilemodeling.com/index.htm, Last accessed Sep 10, 2007.

Atkinson, G., Hagemeister, J., Oman, P. and Baburaj, A., Directing software development projects with product metrics. *Proc. 5th. International Symposium on Software Metrics*, (1998), 193-204.

Barbosa, S.D.J. and de Paula, M.G., Interaction modeling as a binding thread in the software development process. *Proc. Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction at International Conference on Software Engineering (ICSE '03)*, (2003), 84-91.

Barghouti, N.S., Supporting cooperation in the Marvel process-centered SDE. *Proc. Fifth Software Engineering Symposium on Practical Software Development Environments*, (1992), 21 - 31.

Barnes, J. and Leventhal, L., Turning the tables: Introducing software engineering concepts in a user interface design course. *Proc. 32nd SIGCSE Technical Symposium on Computer Science Education*, (2001), 214-218.

Basili, V.R. and Weiss, D., A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, Vol.*SE-10* (6), (1984), 728-738.

Basili, V.R. and Rombach, H.D., The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, Vol.*14* (6), (1988), 758-753.

Basili, V.R., Briand, L.C. and Melo, W.L., A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, Vol.*22* (10), (1996), 751-761.

Bass, L.J. and John, B.E., Supporting Usability Through Software Architecture. *IEEE Computer*, Vol.*34* (10), (2001), 113-115.

Bass, L.J. and John, B.E., Linking usability to software architecture patterns through general scenarios. *The Journal of Systems and Software*, Vol.*66*, (2003), 187-197.

Beck, K., Embracing change with extreme programming. *IEEE Software*, Vol.*32* (10), (1999), 70-77.

Beck, K., *Extreme programming explained: Embrace change*. Addison-Wesley, 2000.

Bevan, N. and Azuma, M., Quality in use: incorporating human factors into the software engineering lifecycle. *Proc. Third IEEE International Software Engineering Standards Symposium and Forum: 'Emerging International Standards'*, (1997), 169-179.

Bevan, N. and Bogomolni, I., Incorporating user quality requirements in the software development process. *Proc. 4th International Software & Internet Quality Week Conference (QWE2000)*, (2000).

Bharat, K.A. and Hudson, S.E., Supporting distributed, concurrent, one-way constraints in user interface applications. *Proc. 8th annual ACM symposium on user interface and software technology (UIST)*, (1995), 121-132.

Blomberg, J.L., Ethnography: Aligning field studies of work and system design. in Monk, A.F. and Gilbert, N. eds. *Perspectives on HCI: Diverse approaches*, Academic Press, London, (1995), 175-198.

Boehm, B.W., *Software Engineering Economics*. Prentice-Hall, Inc., Englewood Cliffs, 1981.

Boehm, B.W., Verifying and validating software requirements and design specifications. *IEEE Software*, Vol.*1* (1), (1984), 75-88.

Boehm, B.W., A spiral model of software development and enhancement. *IEEE Computer*, Vol.*21* (5), (1988), 61-72.

Boehm, B.W., Get ready for agile methods with care. *IEEE Computer*, Vol.*35* (1), (2002), 64-69.

Borning, A. and Duisberg, R., Constraint-based tools for building user interfaces. *ACM Transactions on Graphics (TOG)*, Vol.*5* (4), (1986), 345-374.

Bosch, J., *Design and use of software architectures: Adopting and evolving a product line approach*. Pearson Education, Addison-Wesley, 2000.

Bridging the gaps between software engineering and human-computer interaction. *Proc. International Conference on Software Engineering 2003*, (2003a).

Bruegge, B. and Dutoit, A.H., Communication metrics for software development. *Proc. 19th International Conference on Software Engineering*, (1997), 271-281.

Bush, V., As we may think. *Atlantic Monthly*, Vol.*176*, (1945), 101-108.

Card, S.K., Moran, T.P. and Newell, A., The keystroke-level model for user performance time with interactive systems. *Communications of ACM*, Vol.*23* (7), (1980), 396-410.

Card, S.K., Newell, A. and Moran, T.P., *The psychology of human-computer interaction*. L. Erlbaum Associates, 1983.

Carroll, J.M., Rosson, M.B., Chin Jr., G. and Koenemann, J., Requirements development in scenario-based design. *IEEE Transactions on Software Engineering*, Vol.*24* (12), (1998), 1156-1170.

Chidamber, S.R. and Kemerer, C.F., A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, Vol.*20* (6), (1994), 476-493.

Chok, S.S. and Marriott, K., Automatic construction of user interfaces from constraint multiset grammars. *Proc. 11th IEEE International Symposium on Visual Languages*, (1995), 242-249.

Click & Pledge, Click And Pledge, (2007), http://www.clickandpledge.com/, Last accessed Sep 10, 2007.

Closing the Gaps: Software Engineering and Human-Computer Interaction. *Proc. Interact 2003 Workshop*, (2003b).

Constantine, L.L., Essential Modeling: Use cases for user interfaces. *Interactions*, Vol.*2* (2), (1995), 34-46.

Constantine, L.L. and Lockwood, L.A.D., *Software for use: A practical guide to the models and methods of usage-centered design*. Addison Wesley Longman, Inc., 1999.

Cook, J.E., Votta, L.G. and Wolf, A.L., Cost-effective analysis of in-place software processes. *IEEE Transactions on Software Engineering*, Vol.*24* (8), (1998), 650-663.

Cook, T.D. and Campbell, D.T., *Quasi-experimentation: Design and analysis issues for field settings*. Houghton Mifflin Company, 1979.

da Silva, P.P. and Paton, N.W., Improving UML support for user interface design: A metric assessment of UMLi. *Proc. Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction at International Conference on Software Engineering (ICSE '03)*, (2003), 76-83.

DeMarco, T. and Boehm, B.W., The Agile Methods Fray. *IEEE Computer*, Vol.*35* (6), (2002), 90-92.

Deursen, A.v., Customer involvement in extreme programming. *Proc. Workshop On Customer Involvement (WCI'2001)*, (2001).

Dix, A., Finlay, J., Abowd, G.D. and Beale, R., *Human-Computer Interaction* Pearson Prentice Hall, 2004.

Douglas, S., Tremaine, M., Leventhal, L., Wills, C.E. and Manaris, B., Incorporating human-computer interaction into the undergraduate computer science curriculum. *Proc. 33rd SIGCSE Technical Symposium on Computer Science Education Conference Proceedings*, (2002), 211-212.

Draper, S.W. and Norman, D.A., Software Engineering for User Interfaces. *IEEE Transactions on Software Engineering*, Vol.*SE-11* (3), (1985), 252-258.

Duncan, A.S., Software development productivity tools and metrics. *Proc. The 10th International Conference on Software Engineering*, (1988), 41-48.

Duncan, R., The quality of requirements in extreme programming. *CROSS TALK The journal of defense software engineering*, Vol.*June*, (2001), 19-22.

Ege, R.K., Constraint-based user interfaces for simulations. *Proc. 20th conference on winter simulation*, (1988), 263-271.

Emam, K.E., Moukheiber, N. and Madhavji, N.H., An empirical evaluation of the G/Q/M method. *Proc. The 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1*, (1993), 265-289.

Facebook, (2007a), http://www.facebook.com/, Last accessed Sep 10, 2007.

Faculty ONLINE: The information source for higher education, (2007b), http://www.facultyonline.com, Last accessed Sep 10, 2007.

Fenton, N.E. and Pfleeger, S.L., *Software metrics: A rigorous and practical approach*. International Thomson Computer Press, 1997.

Fenton, N.E. and Neil, M., Software metrics: roadmap. *Proc. Conference on The Future of Software Engineering*, (2000), 357-370.

Ferre, X., Juristo, N., Windl, H. and Constantine, L.L., Usability basics for software developers. *IEEE Software*, Vol.*18* (1), (2001), 22-29.

Ferre, X., Integration of Usability Techniques into the Software Development Process. *Proc. Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction at International Conference on Software Engineering (ICSE '03)*, (2003), 28-35.

Forbus, K.D., Qualitative process theory. in Bobrow, D.G. ed. *Qualitative reasoning about physical systems*, The MIT Press, (1985), 85-168.

Glass, R.L., Extreme programming: the good, the bad, and the bottom line. *IEEE Software*, Vol.*18* (6), (2001), 111-112.

Hanson, W., User engineering principles for interactive systems. *Proc. AFIPS Conference Proceedings 39, Fall Joint Computer Conference*, (1971), AFIPS Press, 523-532.

Hartson, H.R. and Hix, D., Toward empirically derived methodologies and tools for human-computer interface development. *International Journal of Man-Machine Studies*, Vol.*31*, (1989), 477-494.

Henderson-Sellers, B., OO software process improvement with metrics. *Proc. Sixth IEEE International Symposium on Software Metrics*, (1999), 2-8.

Herbsleb, J.D. and Grinter, R.E., Conceptual simplicity meets organizational complexity: Case study of a corporate metrics program. *Proc. The 20th International Conference on Software Engineering*, (1998), 271-280.

Hix, D. and Hartson, H.R., *Developing user interfaces: Ensuring usability through product & process*. John Wiley & Sons, Inc., 1993.

Horrian, H., Mahmud, S. and Karthikeyan, S., Requirements engineering in agile methods, (2003), http://sern.ucalgary.ca/courses/CPSC/601.93/F2003/papers/AgileRE.pdf, Last accessed Sep 10, 2007.

IEEE recommended practice for software requirements specifications *IEEE Std 830 Software engineering standards committee of the IEEE computer society*, (1998).

IEEE/EIA 12207.1 Software life cycle processes--Life cycle data *Standard for Information Technology*, (1997).

Information Processing Systems--Open Systems Interconnection--LOTOS--A Formal Description Technique Based on the Temporal Ordering of Observational Behavior *ISO 8807*, (1989).

Joeris, G., Change management needs integrated process and configuration management. *Proc. 6th European conference held jointly with the 5th ACM SIGSOFT*, (1997), 125--141.

Juristo, N., Lopez, M., Moreno, A.M. and Sanchez, M.I., Improving software usability through architectural patterns. *Proc. Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction at International Conference on Software Engineering (ICSE '03)*, (2003), 28-35.

Kääriäinen, J., Koskela, J., Takalo, J., Abrahamsson, P. and Kolehmainen, K., Supporting requirements engineering in extreme programming: managing the user stories. *Proc. 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA 2003)*, (2003).

Krasner, G.E. and Pope, S.T., A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, Vol.*1* (3), (1988), 26-49.

Krauskopf, R. and Rash, F., Independent verification and validation. *IEEE Potentials*, Vol.*9* (2), (1990), 12-14.

Kwaiter, G., Gaildrat, V. and Caubet, R., Modelling with constraints: a bibliographical survey. *Proc. 1998 IEEE Conference on Information Visualization*, (1998), 211 - 220.

Landauer, T.K., *The trouble with computers. Usefulness, usability, and productivity*. The MIT Press, London, England, 1995.

Latzina, M. and Rummel, B., Soft(ware) skills in context: Corporate usability training aiming at cross-disciplinary collaboration. *Proc. 16th Conference on Software Engineering Education and Training*, (2003), 52-57.

Leonardi, M.C. and Leite, J.C.S.d.P., Using business rules in extreme requirements. *Proc. The Fourteenth International Conference on Advanced Information Systems Engineering (CAiSE'02)*, (2002), Springer-Verlag, 420-435.

Leventhal, L. and Barnes, J., Two for one: Squeezing human-computer interaction and software engineering into a core computer science course. *Computer Science Education*, Vol.*13* (3), (2003), 177-190.

Lewis, R.O., *Independent verification and validation: A life cycle engineering process for quality software*. John Wiley & Sons, Inc, 1992.

Li, W., Software product metrics. *IEEE Potentials*, Vol.*18* (5), (Dec 1999-Jan 2000), 24-27.

Licklider, J.C.R., Man-computer symbiosis. *IRE Transaction on Human Factors in Electronics*, Vol.*HFE-1*, (1965), 4-11.

MailTags, MailTags 2.0, (2007), http://www.indev.ca/MailTags.html, Last accessed Sep 10, 2007.

Mantei, M.M. and Teorey, T.J., Incorporating behavioral techniques into the systems development life cycle. *Management Information System (MIS) Quarterly*, Vol.*13* (3), (1989), 257-276.

Matejka, R.M. and Lagnese, T.J., A representational language for qualitative process control. *Proc. 1st international conference on Industrial and engineering applications of artificial intelligence and expert systems* (1998), Vol.1, 475 - 482

Mayhew, D.J., *Principles and guidelines in software user interface design*. Prentice Hall, Edgewood Cliffs, NJ, 1992.

McCauley, R., Resources for teaching and learning about human-computer interaction. *Inroads; ACM SIGCSE Bulletin*, *35* (2), (2003), 16-17.

Mugridge, W.B., Hosking, J.G. and Grundy, J., Towards a constructor kit for visual notations. *Proc. Sixth Australian Conference on Computer-Human Interaction*, (1996), 169-176.

Myers, B.A. and Rosson, M.B., Survey on user interface programming. *Proc. CHI '92*, (1992), 195-202.

MySpace: a place for friends, (2007c), http://www.myspace.com/, Last accessed Sep 10, 2007.

Newman, W.M. and Lamming, M.G., *Interactive System Design*. Addison-Wesley, 1995.

Nielsen, J., *Usability Engineering*. Academic Press, Cambridge, MA, 1993.

Norman, D.A., *The psychology of everyday things*. Basic Books, New York, 1988.

Orkut, (2007d), www.orkut.com, Last accessed Sep 10, 2007.Newell, A. and Simon, H.A., *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

Paetsch, F., Eberlein, A. and Maurer, F., Requirements engineering and agile software development. *Proc. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '03)*, (2003), 308-313.

Parsons, H., What happened at Hawthorne? *Science*, Vol.*183*, (1974), 922-932.

Paulk, M.C., Curtis, B., Chrissis, M.B. and Weber, C., Capability Maturity Model for Software, Version 1.1, Software Engineering Institute, Carnegie Mellon University (CMU/SEI-93-TR-24), Pittsburgh, PA, (1993).

Pawar, S.A., A common software development framework for coordinating usability engineering and software engineering activities, Masters Thesis, *Department of Computer Science*, Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, (2004).

Pew, R.W., Evolution of Human-Computer Interaction: From MEMEX to Bluetooth and beyond. in Jacko, J.A. and Sears, A. eds. *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, Lawrence Erlbaum Associates, Inc., (2003), 1-17.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. and Carey, T., *Human-Computer Interaction*. Addison Wesley, Wokingham, UK, 1994.

Pressman, R.S., *Software engineering: A practitioner's approach*. McGraw-Hill, 2001.

Purao, S. and Vaishnavi, V., Product metrics for object-oriented systems. *ACM Computing Surveys (CSUR)*, Vol.*35* (2), (2003), 191-221.

Pyla, P.S., Pérez-Quiñones, M.A., Arthur, J.D. and Hartson, H.R., Towards a model-based framework for integrating usability and software engineering life cycles. *Proc. Interact 2003 Workshop on "Closing the Gaps: Software Engineering and Human Computer*

Interaction", (2004a), Université catholique de Louvain, Institut d' Administration et de Gestion (IAG) on behalf of the International Federation for Information Processing (IFIP), 67-74.

Pyla, P.S., Pérez-Quiñones, M.A., Arthur, J.D. and Hartson, H.R., What we should teach, but don't: Proposal for a cross pollinated HCI-SE curriculum. *Proc. Frontiers in Education (FIE) Conference*, (2004b), S1H17-22.

Pyla, P.S., Pérez-Quiñones, M.A., Arthur, J.D. and Hartson, H.R., Ripple: An event driven design representation framework for integrating usability and software engineering life cycles. in Seffah, A., Gulliksen, J. and Desmarais, M. eds. *Human-centered software engineering: Integrating usability in the software development lifecycle*, Springer, (2005), 245-265.

Rakitin, S.R., *Software Verification and Validation for Practitioners and Managers*. Boston Artech House, Inc., 2001.

Rosson, M.B. and Carroll, J.M., Integrating task and software development for object-oriented applications. *Proc. SIGCHI conference on Human factors in computing systems*, (1995), 377-384.

Rosson, M.B., Integrating development of task and object models. *Communications of ACM*, Vol.*42* (1), (1999), 49-56.

Rosson, M.B. and Carroll, J.M., *Usability Engineering: Scenario-based development of human-computer interaction*. Morgan Kaufman, San Francisco, 2002.

Royce, W.W., Managing the development of large scale software systems. *Proc. IEEE Western Electronic Show and Convention (WESCON) Technical Papers (Reprinted in Proceedings of the Ninth International Conference on Software Engineering, Pittsburgh, ACM Press, 1989, pp.328--338)*, (1970), Vol.14 (Session A), A/1 1-9.

Seffah, A., Learning the ropes: Human-centered design skills and patterns for software engineers' education. *Interactions*, Vol.*10* (5), (2003), 36-45.

Selby, R.W., Porter, A.A., Schmidt, D.C. and Berney, J., Metric-driven analysis and feedback systems for enabling empirically guided software development. *Proc. 13th International Conference on Software Engineering*, (1991), 288-298.

Shepard, T., Sibbald, S. and Wortley, C., A visual software process language. *Communications of ACM*, Vol.*35* (4), (1992), 37-44.

Shepperd, M., An evaluation of software product metrics. *Information and Software Technology*, Vol.*30* (3), (1988), 177-188.

Shepperd, M., Products, processes and metrics. *Information and Software Technology*, Vol.*34* (10), (1992), 674-680.

Shneiderman, B., Direct manipulation: A step beyond programming languages. *IEEE Computer*, Vol.*16* (8), (1983), 57-62.

Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley Longman, Inc., 1998.

Sousa, K.S. and Furtado, E., RUPi - A unified process that integrates human-computer interaction and software engineering. *Proc. Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction at International Conference on Software Engineering (ICSE '03)*, (2003), 41-48.

Szekely, P. and Myers, B.A., A user interface toolkit based on graphical objects and constraints. *Proc. Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '88)*, (1988), 36-45.

Tate, G. and Verner, J., Software metrics for CASE development. *Proc. Fifteenth Annual International Computer Software and Applications Conference*, (1991), 565-570.

The Joint Task Force on Computing Curricula, Software engineering 2004 - Curriculum guidelines for undergraduate degree programs in software engineering *Computing Curricula Series*, IEEE Computer Society and Association for Computing Machinery, (2004).

The Standish Group, The CHAOS Report, (1994).

The Standish Group, Unfinished Voyages. A follow-up to The CHAOS Report, (1995).

The Standish Group, CHAOS: A Recipe for Success, (1999).

The Standish Group, Extreme CHAOS, (2001).

Usability and Software Engineering Cross-Pollination: The Role of Patterns. *Proc. Interact 2003 Workshop*, (2003c).

Veer, G.V.d. and Vliet, H.V., The human-computer interface *is* the system: A plea for a poor man's HCI component in software engineering curricula. *Proc. 14th Conference on Software Engineering Education and Training*, (2001), 276-288.

Wahl, N.J., Student-run usability testing. *Proc. 13th Conference on Software Engineering Education and Training*, (2000), 123-131.

Wallace, D.R. and Fujii, R.U., Software verification and validation: An overview. *IEEE Software*, Vol.*6* (3), (1989), 10-17.

Weiser, M., The computer for the 21st century. *Scientific American*, *265* (3), (1991), 66-75.

Weiser, M., Some computer science issues in ubiquitous computing. *Communications of ACM*, Vol.*36* (7), (1993), 75-84.

Weiser, M., The world is not a desktop. *Interactions*, Vol.*1* (1), (1994), 7-8.

Widmaier, J., Smidts, C. and Huang, X., Producing more reliable software: mature software engineering process vs. state-of-the-art technology? *Proc. The 22nd International Conference on Software Engineering*, (2000), 88-93.

Appendix A: SE Demographic Survey

CS 5704: Software Engineering Demographic Survey

Please fill out this questionnaire if you wish to be placed on a CS5704 project development team. This information will be used to place you in a team with as balanced a make-up as possible.

Your name (last, first)_____
Your major_____Email_____

Dropping this class will handicap your project team and prevent another student who is interested in taking this class. To the best of your knowledge, are you now 100% committed to staying in this course and participating in a project development team? _____yes _____no

This particular offering of CS5704 is coordinated with CS/ISE5714 (Usability Engineering) and will involve an external research study to collect data concerning communication among students (including Usability Engineering students) for the team project. This aspect of the course will not affect your learning opportunities or your grades. While we would appreciate your cooperation in this study, we offer the chance to opt out and not participate. Opting out also will not affect your grades. Please indicate your choice by checking one of the lines below:
____ I agree to participate in the communication study.
____ I wish to opt out from the communication study.

I will use the following demographic information to assign you to a team, so that each team will have a good balance of skills and backgrounds. Please mark the point on each scale that best represents you.

Programming and software implementation skills:
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Language(s) most proficient (e.g., Java): _____
_____

Real-world experience in software development:
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Specify number of years/ nature of job: _____
_____

Previous human factors and/or HCI education/experience:
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Specify nature of education/ experience: _____
_____

Writing, especially technical writing, skills (in English!):
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Other knowledge, skills, attributes you have that are relevant to a software development project:

Anything else I should know regarding your placement on a team:

Are you officially registered for this course yet? If not, what is your situation?

Signature (confirming above information) _____

Appendix B: UE Demographic Survey

CS/ISE 5714: Usability Engineering Demographic Survey

Please fill out this questionnaire if you wish to be placed on a CS5714 project development team. This information will be used to place you in a team with as balanced a make-up as possible.

Your name (last, first)_____
Your major_____ Email_____

To the best of your knowledge, are you now 100% committed to staying in this course and participating in a project development team? _____yes _____no

This particular offering of CS/ISE5714 is coordinated with CS5704 (Software Engineering) and will involve an external research study to collect data concerning communication among students (including Software Engineering students) for the team project. This aspect of the course will not affect your learning opportunities or your grades. While we would appreciate your cooperation in this study, we offer the chance to opt out and not participate. Opting out also will not affect your grades. Please indicate your choice by checking one of the lines below:
____ I agree to participate in the communication study.
____ I wish to opt out from the communication study.

I will use the following demographic information to assign you to a team, so that each team will have a good balance of skills and backgrounds. Please mark the point on each scale that best represents you.

Previous human factors and/or HCI education/experience:
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Programming and software implementation (e.g., Java) skills:
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Real-world experience in software development:
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Writing, especially technical writing, skills (in English!):
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
None WeakAverage/Medium Strong Very strong

Other knowledge, skills, attributes you have that are relevant to an interface development project:

Anything else I should know regarding your placement on a team:

Are you officially registered for this course yet? If not, what is your situation?

Signature (confirming above information) _____

Appendix C: Questionnaire Pertaining to Project Experience

Team number: _____

Background information

1. What role did you play in this project? SE UE Both (C teams)

2. Please characterize your ***HCI/human factors*** experience, **prior to your experience in this class.**

- **Professional background –** real-world and other usability experience (circle one):

None	Weak	Average/Medium	Strong	Very strong
Did not know what HCI was	Knew what HCI was about but not many details	Knowledgeable about HCI and processes, used HCI principles in a project or two	Knowledgeable with 1-2 years of experience in an HCI-related job	Expert with 3 or more years experience in an HCI-related job

- **Educational background** (circle one):

None	Weak	Average/Medium	Strong	Very strong
Did not know what HCI was	Knew what HCI was about but not many details	Took an introductory undergraduate HCI class	Took more than one HCI class that had semester-long projects	Had many HCI classes, research work is HCI-related

3. Please characterize your ***software engineering*** experience, **prior to your experience in this class.**

- **Professional background –** real-world and other software development experience (circle one):

None	Weak	Average/Medium	Strong	Very strong
Did not know what SE was	Knew what SE was about but not many details	Knowledgeable about SE and processes, used SE principles in a project or two	Knowledgeable with 1-2 years of experience in software development	Expert with 3 or more years experience in software development

- **Educational background** (circle one):

None	Weak	Average/Medium	Strong	Very strong
Did not know what SE was	Knew what SE was about but not many details	Took an introductory undergraduate SE class	Took more than one SE class that had semester-long projects	Had many SE classes, research work is SE-related

4. Please characterize your **programming and software implementation experience** (e.g., C#, Java, web development).

None	Weak	Average/Medium	Strong	Very strong
No programming skills	Knew some programming but never used in a project	Knew programming and used in one or two small projects	Knew programming and used in more than three projects	Professional programmer, do this (or have done this) for a living

Taking into account your experiences associated with your class project in working with your counterpart role in this study (or being the counterpart role yourself if you are a C or D team member), please answer the following questions.

A teams, please answer these questions with respect to **your direct experiences** (and conjecture where applicable) on getting coordination and communication messages and other forms of help (all the email messages you got over the semester directing you to attend your counterpart role's meetings, get your particular work products checked for consistency, etc.).

B teams, please answer these questions based on **conjectures** and your **experience of not getting help** A teams got. If you feel there are particular cases where certain aspects discussed here would have helped (e.g. getting a message like this would have helped prevent such and such) please describe them here.

C and D teams, please answer these questions based **on your observations this semester**, and your own personal experiences working on this project as a C or D team. Most of your comments will be pure conjecture but your best guesses will still be very valuable.

Issues with SE and UE connections

5. Based on the type of your team (A, B, C, D), how aware were you about your **counterpart role's goals** for the system? (**Circle a number rating**)

Aware→ 1 2 3 4 5 ←Not aware

6. Based on the type of your team (A, B, C, D), how aware were you about your **counterpart role's project status** (e.g. how much has been accomplished at each point in time) for the system? (**Circle a number)**

Aware→ 1 2 3 4 5 ←Not aware

7. What percentage of the work done by you and your counterpart role do you think **overlaps** (to the extent that it would save time and resources if done together as a single group like the C team)?

Less than 15% 15-40% 40-60% 60-85% greater than 85%

8. To what extent did your previous professional/other experience **interfere with** your designated role?

Highly interfered Somewhat interfered No effect

9. Did your previous professional/other experience **support** your designated role?

Greatly helped Helped Neutral Did not help that much Did not help at all

10. Were there any **conflicts** with your counterpart role because of your previous professional/other experience?

Many conflicts Some conflicts No conflicts Previous experience helped prevent conflicts

11. How did your previous professional/other experience help in understanding your counterpart role's responsibilities and duties (select on from each row below)?

Greatly helped	Helped	Neutral	Did not help that much	Did not help at all
I was able to appreciate their expertise, skill, work load, design challenges, etc.			I was frustrated they were doing such a poor job, I could have done better	

12. Having a project leader to oversee the two roles and **enforcing the design suggestions of each role** would be:

Very useful Useful Will probably not make any difference Harmful Very Harmful

13. As you interacted with your counterpart roles, how did the following factors affect your role?

- Differences in **terminology**.

Very confusing Confusing There were no differences Easy to understand

- Differences in **iterativeness**.

Very frustrating Frustrating Neutral Satisfying Very Satisfying

- Susceptibility to **change**.

Very frustrating Frustrating Neutral Satisfying Very Satisfying

- Differences in the way each role **represents their designs/work products**.

Problematic Did not have an effect There were no differences Affected positively

14. How often did your counterpart role **influence** your design?

Very often Often Few times Rarely Never

15. With respect to **improving the quality** of your overall system, your counterpart role's influence was:

Very positive Positive Did not influence either way Detrimental Very Detrimental

Communication factors and experiences

16. Based on the type of your team (A, B, C, D), how would you rate **having effective communication** between the two roles with respect to maintaining the quality of the **product** (software system you created)?

Very positive Positive Did not influence either way Detrimental Very Detrimental

17. Based on the type of your team (A, B, C, D), how would you rate **having effective communication** between the two roles with respect to maintaining the quality of the **development process**?

Very positive Positive Did not influence either way Detrimental Very Detrimental

18. Having communication between the SE and UE roles as provided to A teams is **important.**

Strongly disagree Disagree Neutral Agree Strongly Agree

19. Having communication between the SE and UE roles as provided to A teams is **useful**.

Strongly disagree Disagree Neutral Agree Strongly Agree

20. Having communication between the SE and UE roles as provided to A teams **unduly complicates** the total process.

Strongly disagree Disagree Neutral Agree Strongly Agree

21. Having communication between the SE and UE roles as provided to A teams **increases overall workload** for both roles.

Strongly disagree Disagree Neutral Agree Strongly Agree

22. **Lack of communication** between the SE and UE roles as experienced by B teams **creates serious problems** for the overall project effort.

Strongly disagree Disagree Neutral Agree Strongly Agree

23. **Lack of communication** between the SE and UE roles as experienced by B teams **increases overall workload** for both roles at the end.

Strongly disagree Disagree Neutral Agree Strongly Agree

24. The overall quality of the **product** created from B teams would have improved if they were provided with the communication that was afforded to A teams.

Strongly disagree Disagree Neutral Agree Strongly Agree

25. The overall quality of the **process** as experienced by B teams would have improved if they were provided with the communication that was afforded to A teams

Strongly disagree Disagree Neutral Agree Strongly Agree

26. Having **group email IDs** for the SE and UE sub-teams helps with communication of related concerns (e.g. the sub-team was able to be aware of what the current status of their part of the project was).

Strongly disagree Disagree Neutral Agree Strongly Agree

27. Having a **combined email ID** that includes SE and UE roles helps with communication of concerns relating to the total system.

Strongly disagree Disagree Neutral Agree Strongly Agree

Team factors and experiences

28. If we were to repeat this study again and you were given a choice to pick a particular group to be in, what would you pick? Answer this question with respect to overall quality of the system not amount of work involved.

A team	B team	C team	D team
Have a counterpart role with communication	Without communication	Play both roles yourself	No counterpart team, only SE or UE

Briefly explain why:

29. Based on the type of your team (A, B, C, D), how would you rate your **overall experience** working on this project?

Very frustrating Frustrating Neutral Satisfying Very Satisfying

30. Being a dual expert like those in C team **increases overall workload** for the overall project effort.

Strongly disagree Disagree Neutral Agree Strongly Agree

31. Being a dual expert like those in C team **helps in having a broader and richer design** for overall system.

Strongly disagree Disagree Neutral Agree Strongly Agree

32. Being in a D team (without a counterpart team) **helps in being creative** with the design without worrying about **feasibility, implementation, or other constraints**.

Strongly disagree Disagree Neutral Agree Strongly Agree

33. Being in a D team (without a counterpart team) **has the least amount of work** as they did not have to worry about the counterpart team.

Strongly disagree Disagree Neutral Agree Strongly Agree

34. In this study each role had **different deliverables and timelines** for various phases of project. This was:

Very problematic Problematic Did not affect the quality of system Good Very good

35. Having **combined project deliverables** (e.g. SE's requirements plus UE's ethnography as single report) would have helped achieve a more cohesive and higher quality system than having them as separate documents.

Strongly disagree Disagree Neutral Agree Strongly Agree

36. Having **combined project grades** (e.g. one grade for "requirements" which includes SE's requirements and UE's ethnography) would have helped achieve a more cohesive and higher quality system than having them graded separately.

Strongly disagree Disagree Neutral Agree Strongly Agree

37. How would you rate your counterpart team's **performance** on this project?

Excellent Good Fair Poor Horrendous

38. How would you rate your **experience** working with your counterpart team?

Enlightening Informative Did not learn anything new Waste of time

Coordination factors and experiences

39. When you are doing an activity relevant to your role (e.g. UE doing ethnography or SE doing requirements engineering), **having a representative** from the other team is **important**.

Strongly disagree Disagree Neutral Agree Strongly Agree

40. When you are doing an activity relevant to your role (e.g. UE doing ethnography or SE doing requirements engineering), **having a representative** from the other team is **useful**.

Strongly disagree Disagree Neutral Agree Strongly Agree

41. When you are doing an activity relevant to your role, **having a representative** from the other team **provides insights** to the overall project that we would have missed otherwise.

Strongly disagree Disagree Neutral Agree Strongly Agree

42. When you are doing an activity relevant to your role, **having a representative** from the other team provides a better understanding of **what is feasible** (constraints between SE and UE designs).

Strongly disagree Disagree Neutral Agree Strongly Agree

43. **Getting messages informing you** about your counterpart team's activities and suggesting you have a representative participate is **important.**

Strongly disagree Disagree Neutral Agree Strongly Agree

44. **Getting messages informing you** about your counterpart team's activities and suggesting you have a representative participate is **useful.**

Strongly disagree Disagree Neutral Agree Strongly Agree

Constraint and dependency checks and experiences

45. Making sure that the design artifacts from the SE and UE roles are **consistent** is **important.**

Strongly disagree Disagree Neutral Agree Strongly Agree

46. Making sure that the design artifacts from the SE and UE roles are **consistent** is **useful.**

Strongly disagree Disagree Neutral Agree Strongly Agree

47. **Getting periodic messages** with specific directives on consistency of design artifacts between the two roles is **important** (e.g. Message saying "make sure all SE use cases are supported by the UE usage scenarios").

Strongly disagree Disagree Neutral Agree Strongly Agree

48. **Getting periodic messages** with specific directives on consistency of design artifacts between the two roles is **useful** (e.g. Message saying "make sure all SE use cases are supported by the UE usage scenarios").

Strongly disagree Disagree Neutral Agree Strongly Agree

49. Having **periodic checkpoints** where both roles sit together and make sure they are in sync is important

Strongly disagree Disagree Neutral Agree Strongly Agree

Synchronization issues and experiences

50. How often did you have to **wait for your counterpart role's work products** before you could proceed with your work activity?

Very often Often Sometimes Rarely Never

51. How did such delay waiting for your counterpart role's work products **affect your work activities**?

Was highly detrimental Was somewhat detrimental No effect Was actually helpful

52. How often did you encounter situations where you wish you knew what your counterpart role was doing so you could make informed decisions about your own process?

Very often Often Sometimes Rarely Never

53. **Getting periodic messages** with specific directives on what work products need to be ready for the other role is **important** (e.g. Message saying "make sure you have your UE screen designs ready because the SE role is going into implementation next week").

Strongly disagree Disagree Neutral Agree Strongly Agree

54. **Getting periodic messages** with specific directives on what work products need to be ready for the other role is **useful** (e.g. Message saying "make sure you have your UE screen designs ready because the SE role is going into implementation next week").

Strongly disagree Disagree Neutral Agree Strongly Agree

Other

55. What are the most important lessons you have learned participating in this experiment?

Appendix D: Questionnaire Pertaining to Joint SE-UE Course Experience

Team type (circle one): A B C D

Questionnaire for evaluating joint offering of SE and UE classes

1. Which class were you in? (circle one) **SE** **UE** **Both**

2. Maintaining **personal journals** to record everyday project activities and experiences was **very time consuming**

   Strongly disagree Disagree Neutral Agree Strongly Agree

3. Maintaining **personal journals** to record everyday project activities and experiences was **useful to record the progress of a project**

   Strongly disagree Disagree Neutral Agree Strongly Agree

4. Maintaining **personal journals** to record everyday project activities and experiences **helped in reconstructing** what happened with the project later on

   Strongly disagree Disagree Neutral Agree Strongly Agree

5. Based on my experience this semester, I would **maintain a journal** for all my future projects.

   Strongly disagree Disagree Neutral Agree Strongly Agree

6. Having **group IDs** for the sub-teams and overall teams was helpful

   Strongly disagree Disagree Neutral Agree Strongly Agree

7. Having **group IDs** for teams was **overwhelming** with the amount of communication

   Strongly disagree Disagree Neutral Agree Strongly Agree

8. I would recommend **group IDs** for future group projects

   Strongly disagree Disagree Neutral Agree Strongly Agree

9. Have you taken a class/have **experience in your counterpart team's area** before this semester? (e.g. if you are a UE role, have you taken SE classes or have SE experience)

   Yes No

10. Based on your experience/answer from above question, having a joint SE and UE class like this is **(circle one)** [More valuable Neutral Less valuable] than having the classes offered independently.

11. Overall, I learned more than I would in this class because of this "**connected**" offering of SE and UE classes

    Strongly disagree Disagree Neutral Agree Strongly Agree

12. Overall, I prefer taking a class the regular way that is **not connected** like this

    Strongly disagree Disagree Neutral Agree Strongly Agree

13. Working with **real clients** for the class project made the learning experience better

Strongly disagree Disagree Neutral Agree Strongly Agree

14. Having **structured meetings** with clients as planned/arranged by the GTA was better than having to contact them ourselves

    Strongly disagree Disagree Neutral Agree Strongly Agree

15. Having real clients provided a more **realistic understanding** of the requirements for the system

    Strongly disagree Disagree Neutral Agree Strongly Agree

Were there any **problems** with having real clients? Briefly explain:

16. Having a **single set of deliverables** for both SE and UE would have made the learning experience better

    Strongly disagree Disagree Neutral Agree Strongly Agree

17. Having a **symposium** at the end of the semester helped learn from others' experiences

    Strongly disagree Disagree Neutral Agree Strongly Agree

18. Based on this experience, I would **prefer** courses which have an end-of-semester symposium

    Strongly disagree Disagree Neutral Agree Strongly Agree

19. Having the class follow the **entire development life cycle** from requirements to evaluation afforded a better learning experience than classes with projects which go only a part of the way.

    Strongly disagree Disagree Neutral Agree Strongly Agree

20. (This question for SE students only) Having **constraints on the development platform** helped
Strongly disagree Disagree Neutral Agree Strongly Agree

    Any comments:

21. Based on the type of your team (A, B, C, D) how would you rate your **learning experience** in this class (as opposed to what you perceive members from other teams learned from their project)? (circle a number)

    Best→ 1 2 3 4 5 ←Worst

22. Based on your experience in this unique offering of the two classes, what can be **done differently** to improve such an offering the next time?

Appendix E: IRB Informed Consent and Approval

Informed Consent Form

Virginia Polytechnic Institute and State University

Informed Consent for Participant of Investigative Project

Title of Project: **Evaluation of the Ripple Software Development Framework**

Investigators: **H. Rex Hartson and Pardha S. Pyla**

I. THE PURPOSE OF THIS RESEARCH

As described in the presentation by Pardha S. Pyla, the objective of this study is to evaluate the Ripple framework.

II. PROCEDURES

Graduate students from CS5714-Usability Engineering class will be treated as UE experts and CS5704-Software Engineering students will be treated as SE experts. We will form two types of six-member teams: the first type will have students will single expertise where three experts are from the UE class and three from SE class. The second type of team will have dual-experts: students enrolled in both classes. Of all the single-expertise teams, half will use the Ripple framework and the other half will use a framework that is representative of how interactive software is currently developed in the real world. The dual expertise teams will use Ripple framework. All the students in both classes will be required to maintain a journal of the total number of hours spent in working on this project individually, working with teams, and other project related attributes (e.g. number of defects discovered in the project, number of changes resulting due to the functional and user interface components, etc.). Each team will be assigned a group email and will be required to use it for all project related communication. These email exchanges will be analyzed for themes after the end of the semester. The software systems developed will be evaluated at the end of the semester for various quality metrics. At the end of the semester, the students will be asked to fill out questionnaires about their experience in the class. All data collected in this study will be kept annonymous and the students will be informed at the start of the semester about this study. THE GRADES OF THE STUDENTS WILL NOT BE AFFECTED BY PRODUCTS THEY CREATE OR THEIR ANSWERS TO QUESTIONAIRES OR JOURNALS. THEY WILL ONLY BE EVALUATED FOR THE PROCESS THEY USED (WHICH IS THE SAME FOR ALL TYPES OF TEAMS) TO CREATE THESE PRODUCTS.

All information that you help us attain will remain anonymous.

III. RISKS

There are no physical or emotional risks associated with this experiment.

IV. BENEFITS OF THIS PROJECT

Your participation in this project will provide information that will be used to improve the understanding of the advantages of using a framework such as Ripple in developing interactive software systems. No guarantee of benefits has been made to encourage you to participate. You may receive a synopsis summarizing this research when completed.
You are requested to refrain from discussing your experiences with other members outside your assigned team till the end of the semester to prevent any undue biases.

V. EXTENT OF ANONYMITY AND CONFIDENTIALITY

The results of this study will be kept strictly confidential. Your written consent is required for the researchers to release any data identified with you as an individual to anyone other than personnel working on the project. The information you provide will have your name removed and only a subject number will identify you during analyses and any written reports of the research.
Only your replies to study questions will be used in the research. At no time will the direct use of your name or personal data be made. Data will be stored securely and will be made available only in the context of research publications and discussion. No reference will be made in oral or written reports that could link you to the data nor will you ever be identified as a participant in the project. All data gathered will have your name removed and only a user number will identify each user during analyses and any written reports of the research.

VI. COMPENSATION

Your participation is voluntary and unpaid.

VII. FREEDOM TO WITHDRAW

You are free to withdraw from this study **at any time for any reason and without penalty**.

VIII. APPROVAL OF RESEARCH

This research has been approved, as required, by the Institutional Review Board for projects involving human subjects at Virginia Polytechnic Institute and State University, and by the Department of Computer Science.

IX. PARTICIPANT'S RESPONSIBILITIES

I voluntarily agree to participate in this study, and I know of no reason I cannot participate. I will keep the activities and information discussed confidential from students outside my assigned team.

X. PARTICIPANT'S PERMISSION

I have read and understand the informed consent and conditions of this project. I have had all my questions answered. I hereby acknowledge the above and give my voluntary consent for participation in this project. If I participate, I may withdraw at any time without penalty. I agree to abide by the rules of this project

_____ _____

Signature Date

_____ _____

Name (please print)	Contact: Phone/Address/Email (Optional)

Should I have any pertinent questions about this research or its conduct, I may contact:

Investigator	e-mail
Pardha S. Pyla	ppyla@vt.edu

Faculty Advisor	Telephone/e-mail
Dr. H. Rex Hartson	540 231 4857/ hartson@cs.vt.edu

IRB Chair of the Institutional Review Board for the Protection of Human Subjects	Telephone/e-mail
Dr. David Moore	(540) 231-4358/moored@vt.edu,

IRB Approval for Study:

VirginiaTech

Office of Research Compliance
Institutional Review Board
1880 Pratt Drive (0497)
Blacksburg, Virginia 24061
540/231-4991 Fax: 540/231-0959
E-mail: moored@vt.edu
www.irb.vt.edu
FWA00000572(expires 7/20/07)
IRB # is IRB00000667.

DATE: August 9, 2006

MEMORANDUM

TO: H. Rex Hartson
Pardha Pyla

Approval date: 8/9/2006
Continuing Review Due Date:7/25/2007
Expiration Date: 8/8/2007

FROM: David M. Moore

SUBJECT: **IRB Expedited Approval:** "Evaluation of the Ripple Software Development Framework" , IRB # 06-432

This memo is regarding the above-mentioned protocol. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 56.110. As Chair of the Virginia Tech Institutional Review Board, I have granted approval to the study for a period of 12 months, effective August 9, 2006.

As an investigator of human subjects, your responsibilities include the following:

1. Report promptly proposed changes in previously approved human subject research activities to the IRB, including changes to your study forms, procedures and investigators, regardless of how minor. The proposed changes must not be initiated without IRB review and approval, except where necessary to eliminate apparent immediate hazards to the subjects.
2. Report promptly to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.
3. Report promptly to the IRB of the study's closing (i.e., data collecting and data analysis complete at Virginia Tech). If the study is to continue past the expiration date (listed above), investigators must submit a request for continuing review prior to the continuing review due date (listed above). It is the researcher's responsibility to obtained re-approval from the IRB before the study's expiration date.
4. If re-approval is not obtained (unless the study has been reported to the IRB as closed) prior to the expiration date, all activities involving human subjects and data analysis must cease immediately, except where necessary to eliminate apparent immediate hazards to the subjects.

Important:
If you are conducting **federally funded non-exempt research**, this approval letter must state that the IRB has compared the OSP grant application and IRB application and found the documents to be consistent. Otherwise, this approval letter is invalid for OSP to release funds. Visit our website at http://www.irb.vt.edu/pages/newstudy.htm#OSP for further information.

cc: File

IRB Continuation Approval for Further Analyses:

VirginiaTech

Office of Research Compliance
Institutional Review Board
2000 Kraft Drive, Suite 2000 (0497)
Blacksburg, Virginia 24061
540/231-4991 Fax 540/231-0959
e-mail moored@vt.edu
www.irb.vt.edu

FWA00000572(expires 1/20/2010)
IRB # is IRB00000667

DATE: July 10, 2007

MEMORANDUM

TO: H. Rex Hartson
Pardha Pyla
James D. Arthur

FROM: David M. Moore

Approval date: 8/9/2007
Continuing Review Due Date:7/25/2008
Expiration Date: 8/8/2008

SUBJECT: **IRB Expedited Continuation 1:** "Investigating Communication Effects Within Software Development Teams" , IRB # 06-432

This memo is regarding the above referenced protocol which was previously granted expedited approval by the IRB. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 56.110. Pursuant to your request, as Chair of the Virginia Tech Institutional Review Board, I have granted approval for extension of the study for a period of 12 months, effective as of August 9, 2007.

Approval of your research by the IRB provides the appropriate review as required by federal and state laws regarding human subject research. As an investigator of human subjects, your responsibilities include the following:

1. Report promptly proposed changes in previously approved human subject research activities to the IRB, including changes to your study forms, procedures and investigators, regardless of how minor. The proposed changes must not be initiated without IRB review and approval, except where necessary to eliminate apparent immediate hazards to the subjects.
2. Report promptly to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.
3. Report promptly to the IRB of the study's closing (i.e., data collecting and data analysis complete at Virginia Tech). If the study is to continue past the expiration date (listed above), investigators must submit a request for continuing review prior to the continuing review due date (listed above). It is the researcher's responsibility to obtained re-approval from the IRB before the study's expiration date.
4. If re-approval is not obtained (unless the study has been reported to the IRB as closed) prior to the expiration date, all activities involving human subjects and data analysis must cease immediately, except where necessary to eliminate apparent immediate hazards to the subjects.

cc: File

Invent the Future

Appendix F: Sample Journal Entries

Individual hours:

Date	Individual hours	Description
08/29/06	1	Wrote initial overview document
08/30/06	0.5	Edited overview document
…	…	…

Group hours:

Date	Group hours	Team members present	Description
08/28/06	0.75	SE team	Pre-client-meeting meeting
08/28/06	0.25	SE team, client reps	Initial client meeting
…	…	…	…

Unexpected changes:

Date	Brief description of change	Cause for change	Resolution if any	Comments
09/25/06	Clients need to be able to discount plants on POS.	Clients discount plants when they are not selling well or want to get rid of inventory. This was not mentioned before by clients and the team had never questioned the clients about discounts.	Resolved: POS will be added with discount option on each plant and overall discount.	I am not sure if this is a requirement we should have detected earlier or something that clients just remembered. Whose fault is it? or nobodys?
…	…	…	…	…

Issues with SE-UE interaction:

Date	Brief description of issue	Underlying cause	Resolution if any	Comments
10/25/2006	Once again we are having a lot of problems with the SE Team responding our emails in attempts to meeting	Not sure	Respond already.	Once again, becoming a pain to schedule everyone together.
…	…	…	…	…

Other experiences:

Date	Experience	Insight
9/12/2006	***** signed up to do a significant amount of Project 2 saying he has time to do it & he doesn't know how much he can contribute to Project 3	This kind of flexibility is ok with me but makes me a little nervous to rely on just one person to get a bulk of the info done.

Appendix G: Group Project Timetable and Schedule

			Usability Engineering		Software Engineering	
Week #	**Day**	**Date**	**Topic**	**Project Deadlines**	**Topic**	**Project Deadlines**
1	T	22-Aug	Introduction, Life Cycle		Introduction	
	R	24-Aug	Life Cycle, Systems Analysis, product concept statement		SE Models	
2	T	29-Aug	Systems Analysis, exercises		RE Elicitation and Analysis	
	R	31-Aug	Design	**Project 1: Topic, Client, and Product Concept Statement**	Requirements engineering (verification and validation)	
3	T	5-Sep	Scenarios & Screen Design, exercise		Triage	**Project 1: Product overview document**
	R	7-Sep	Scenarios & Screen Design show & tell (in class project), Benchmark Tasks, Usability Specifications		Specification	
4	T	12-Sep			High-level Design (Arch style, Data design, DFD, I/O)	
	R	14-Sep	Rapid Prototyping, exercise	**Project 2: Systems Analysis**		
5	T	19-Sep				**Project 2: Requirements (SRS)**
	R	21-Sep	Usability evaluation (Intro), Usability evaluation (Before data collection), Usability evaluation (During data collection)			**Project 3a: High-level design**
6	T	26-Sep	Usability evaluation (During data collection), Critical incident training exercise (individual), Camtasia screen recording demo		Low-level Design (Component decomposition, Test Description, Document)	
	R	28-Sep	Introduction to the HCI (Usability Methods Lab), DCART UE-support tools demo, Data collection demo (The NY Ticket Kiosk)	**Project 3: Scenarios, Screen Design, and Early Prototype**		
7	T	3-Oct	Usability evaluation (During data collection) team exercise			

	R	5-Oct	Usability evaluation (After data collection), team exercise			**Project 3b: Design document (Both high-level and low-level)**
8	T	10-Oct	In-class exercise Kiosk RP & evaluation show & tell		Advanced topics	
	R	12-Oct	Team (outside) project prototype demos (in-class overview)			
9	T	17-Oct	User Action Framework, design guidelines	**Project 4: High-Fidelity Prototype and Usability Specifications**		
	R	19-Oct				
10	T	24-Oct	Usability guidelines exercises, Usability Inspection			
	R	26-Oct	Web usability			**Project 4: Code deliverable**
11	T	31-Oct	Usability problem analysis			
	R	2-Nov		**Project 5: Formative Usability Evaluation**		
12	T	7-Nov				
	R	9-Nov				
13	T	14-Nov				
	R	16-Nov	Open	**Project 6: Design Iteration**		**Project 6: User acceptance test**
Thanksgiving break						
14	T	28-Nov	Review and Faculty Evaluation (***attendance required***)	**Project 8: Team Member Evaluations**		
	R	30-Nov	**Symposium in 110 KWII 2:00PM to 5:00PM**	**Project 7: Oral Presentations**	**Symposium in 110 KWII 2:00PM to 5:00PM**	**Project 7: Oral Presentations**
15	T	5-Dec	Day OFF			
	~ - ~		11 Dec 7:45 AM: Final exam		08 Dec 7:45AM: Final exam	

Appendix H: Curriculum Vita

PARDHA SARADHI PYLA

Education

- **Doctor of Philosophy,** Computer Science, **Virginia Tech** - Blacksburg, VA Fall 2007
  Advisor: Dr. Rex Hartson ***GPA*: 3.82/4**
  Dissertation: Connecting the usability and software engineering life cycles through a communication-fostering software development framework and cross-pollinated computer science courses
- **Graduate Certificate,** Human-Computer Interaction, **Virginia Tech** - Blacksburg, VA Dec 2006
- **Master of Science,** Computer Science, **Virginia Tech** - Blacksburg, VA May 2006
- **Master of Science,** Computer Engineering, **Virginia Tech** - Blacksburg, VA May 2002
- **Bachelor of Technology,** Electronics & Commn. Engg., **Nagarjuna University**, India May 1999
- **Diploma,** Network-Centered Computing, **NIIT**, India July 1998

Areas of Specialization and Competence

Human-Computer Interaction (HCI), Usability Engineering, Software Engineering and Design

Selected Technical Skills

Development techniques: Ethnographic analysis, business process modeling, user-class analysis, task analysis, design/usage scenarios, screen designs, rapid prototyping, usability evaluation techniques

Design methodologies, frameworks, and architectures: Star life cycle for Usability Engineering, User Action Framework (a structured knowledgebase of usability concepts), Model View Controller (MVC), Scenario-Based Design

Refereed Publications

- Ball, R., Pyla, P.S. and Pérez-Quiñones, M.A. OSI and ET: Originating Source of Information and Evidence Traceability. *Proc. of the CHI 2007 work-in-progress*, 2007
- Pyla, P.S., Hartson, H.R., Arthur, J.D., Smith-Jackson, T.L. and Pérez-Quiñones, M.A. Evaluating Ripple: Experiences from a Cross Pollinated SE-UE Study. *Proc. of the CHI 2007 Workshop on Increasing the Impact of Usability Work in Software Development*, 2007
- Tungare, M., Pyla, P.S., Sampat, M., Pérez-Quiñones, M.A., Syncables: A Framework to Support Seamless Data Migration across Multiple Platforms; *Proc. of the IEEE International Conference on Portable Information Devices,* IEEE Portables 2007.
- Howarth, J., Pyla, P.S., Yost, B., Haciahmetoglu, Y., Young, D., Ball, R., Lambros, S. and Layne, P., Designing a conference for women entering academe in the sciences and engineering. *Advancing Women in Leadership (AWL)*, vol. 24, (2007).
- Pyla, P.S., Pérez-Quiñones, M.A., Arthur, J.D. and Hartson, H.R., Ripple: An event driven design representation framework for integrating usability and software engineering life cycles. In Seffah, A., Gulliksen, J. and Desmarais, M. eds. *Human-centered software engineering: Integrating usability in the software development lifecycle*, Springer, (2005), 245-265**.**

- Tungare, M., Pyla, P.S., Miten, S. and Pérez-Quiñones, M.A. Defragmenting Information with the Syncables Framework. *Proc. of the 2nd Invitational Workshop on Personal Information Management at SIGIR 2006*, (2006).
- Pyla, P.S., Tungare, M. and Pérez-Quiñones, M.A., Multiple user interfaces: Why consistency is not everything, and seamless task migration is key. *Proc. of the CHI 06 Workshop on The Many Faces of Consistency in Cross-Platform Design*, (2006).
- Pyla, P.S., Howarth, J.R., Catanzaro, C. and North, C., Vizability: A Tool for usability engineering process improvement through the visualization of usability problem data. *Proc. of the 44th ACM Southeast Conference (ACMSE)*, (2006), 620-625.
- Tungare, M., Pyla, P.S., Glina, V., Bafna, P., Balli, U., Zheng, W., Yu, X. and Harrison, S., Embodied data objects: Tangible Interfaces to Information Appliances. *Proc. of the 44th ACM Southeast Conference (ACMSE)*, (2006), 359-364.
- Pyla, P.S., Pérez-Quiñones, M.A., Arthur, J.D. and Hartson, H.R., What we should teach, but don't: Proposal for a cross pollinated HCI-SE curriculum. *Proc. of the Frontiers in Education (FIE) Conference*, (2004), S1H17-22.
- Chen, J., Pyla, P.S. and Bowman, D.A., Testbed evaluation of navigation and text display techniques in an information-rich virtual environment. *Proc. of the IEEE Virtual Reality*, (2004), 181-188.
- Bowman, D.A., North, C., Chen, J., Polys, N.F., Pyla, P.S., and Yilmaz, U., Information-rich virtual environment: Theory, tools, and research agenda. *Proc. of the ACM Virtual Reality Software and Technology*, Oct 2003, 81-90.
- Pyla, P.S., Pérez-Quiñones, M.A., Arthur, J.D. and Hartson, H.R., Towards a model-based framework for integrating usability and software engineering life cycles. *Proc. of the Interact 2003 Workshop on "Closing the Gaps: Software Engineering and Human Computer Interaction"*, (2003), Université catholique de Louvain, Institut d' Administration et de Gestion (IAG) on behalf of the International Federation for Information Processing (IFIP), 67-74.

Publications ready for submission/under review

- Tungare, M., Pyla, P.S., Pérez-Quiñones, M.A., and Harrison S., Personal Information Ecosystems and Implications for Design, 2007
- Pérez-Quiñones, M.A., Tungare, M., Pyla, P.S. and Kurdziolek, M., A Special Topics Course on Personal Information Management
- Pyla, P.S., Tungare, M., Holman, J. and Pérez-Quiñones, M.A., Continuous UIs for seamless task migration in MPUIs: Bridging task-disconnects. (*Presented at the Human Computer Interaction Consortium, HCIC 2006*)
- Bohner, S., George, B., Pyla, P.S., and McCrickard, S. Model-Based Evolution for Change Tolerant Web Software.

Professional Distinctions and Activities

- **Outstanding Doctoral Student** for College of Engineering, Virginia Tech, 2007
- **Citizen Scholar Award**, Graduate Student, Virginia Tech, 2007
- **Best In Show** and **Crowd Favorite**, Creative Design Contest, Graduate School, Virginia Tech, 2007
- **First Place, Photography Category,** Graduate Life Center Art and Photography Contest, Graduate School, Virginia Tech, Fall 2007

- Member, **Graduate Student of the Year Selection Committee**, Graduate School, Virginia Tech, 2007
- Recipient of $400 **Graduate Research Development Project** A**ward** from Graduate Student Assembly, Virginia Tech, Fall 2006
- **First Place, Photography Category,** Graduate Life Center Art and Photography Contest, Graduate School, Virginia Tech, Fall 2006
- **First Place in Engineering,** Interactive demo and poster presentation, *Embodied data objects: Tangible Interfaces to Information Appliances*, 22nd Annual Research Symposium and Exposition, Graduate School, Virginia Tech, 2006
- **Technical Reviewer,** CHI (2007), Frontiers in Education (FIE 07), Software Quality Journal (2006), 2nd Latin American Conference on Human-Computer Interaction (2005)
- **Featured Graduate Student of the Month**, Graduate School, Virginia Tech March, 2005
- **Dean's List for Outstanding Instructors/Teaching Excellence** from the College of Engineering, Virginia Tech, Summer 2004
- **Graduate Student Representative to the Faculty,** Dept. of Computer Science, Virginia Tech, 2004
- **Secretary of the Interior,** Graduate Council, Dept. of Computer Science, Virginia Tech, 2004/2005
- **Computer Science Graduate Representative to the Graduate Student Assembly,** Virginia Tech, 2004-2007
- **Graduate Council Mentor** to new graduate students, Dept. of Computer Science, Virginia Tech, 2004

Professional and Academic Experience

SoF Program Coordinator, Dept. of Computer Science, Virginia Tech (Spring 2007)

- Assisting in running a semester-long NSF-funded Scholars of the Future program
- Mentoring eight minority undergraduate students in Computer Science to encourage the pursuit of graduate studies
- Facilitating networking and fostering communication among participating scholars from Virginia Tech and Auburn University
- Planning relevant activities for the scholars to enrich their undergraduate experience

Usability Consultant, Meridium Inc., Roanoke, Virginia (Summer and Winter of 2006)

- Performed usability walkthroughs for Meridium's suite of software systems
- Identified usability problems and suggested redesigns
- Introduced usability best practices and guidelines into the development life cycle

REU Program Coordinator, Center for HCI, Virginia Tech (Summer 2006)

- Assisted in running an eight-week NSF-funded Research Experience for Undergraduates program
- Mentored eight undergraduates from seven universities in various aspects of HCI research
- Led discussions and sometimes lectured on a variety of topics in HCI

Graduate Research Assistant, Dept. of Computer Science, Virginia Tech (Summer 2005)

- Part of three-member research and design team for DCART (Usability Data Collection, Analysis, and Reporting Tool) project

- Designed tools to capture and support usability engineering life cycle activities and products
- Created a conceptual metaphor that uses multi-view visualization to make system model intuitive
- Developed lo-fidelity paper prototypes and conducted lab-based usability evaluations

Instructor, Dept. of Computer Science, Virginia Tech (Summer 2005/2004)

- Prepared and lectured on a wide range of topics in Human-Computer Interaction (Junior/Senior-level undergraduate course)
- Facilitated and led in-class activities to practice various phases of usability engineering process
- Supervised semester-long group projects on developing user interfaces
- Administered and graded tests, projects, and assignments

Graduate Teaching Assistant, Dept. of Computer Science, Virginia Tech

Assisted in teaching, graded assignments and tests, conducted in-class activities, and sometimes lectured:

- **Usability engineering** – Graduate course (Spring 2003/2004, Fall 2004/2006)
- **Human-computer interaction** – Undergraduate course (Summer 2003, Fall 2003/2005, Spring 2006)
- **Graphical user interfaces** – Special study senior-level course (Spring 2005)
- **User interface software** – Special study graduate course (Fall 2002)

Graduate Research Assistant, Dept. of Chemistry, Virginia Tech (June 2001 – May 2002)

- Developed web-based and standalone applications in Java to data-mine, analyze and record patterns in sets of torsion angles of protein structures
- Gathered and analyzed requirements, designed, implemented, and deployed the application for use by researchers in Chemistry

Research Assistant, Dept. of Architecture, Virginia Tech (Mar 2001 – Aug 2002)

Duties involved simulation and optimization of architectural structural data

Project Experience

Continuous User Interfaces (CUIs) for multi-platform user interaction: Currently part of three-member research team addressing the problem of task-disconnect that users face when migrating a task from one platform to another during a task

- Defined task-disconnects (break in continuity that occurs when a user attempts to accomplish a task using more than one computing device) and introduced the concept of CUIs
- Developed a software framework to bridge this task-disconnect, enabling users to seamlessly transition their tasks among different devices
- More than two publications in peer-reviewed conferences (see publications list)

Embodied data objects for tangible interfaces to information appliances: Researched alternative and novel interaction paradigms to mitigate the increasing complexity of computing devices and platforms

- Applied the notion of using tangible everyday objects as embodied representations of active digital media
- Prototyped two service-oriented information appliances (printer and presentation projector) which provide specific "computing" services as dictated by context of interaction

- Designed a framework to support more types of services with little changes to existing setup
- Work published in peer-reviewed conference (see publications list)

Visualization of usability data/metrics for usability process improvement: Started and led a research group that designed and developed Visability, a tool for visualizing usability problem data

- Tool provided problem trends, what-if analysis capability, aggregate analysis, and ability to view recurring issues within usability development life cycle
- Advised and mentored an MS student who took this project as his master's thesis
- Work published in peer-reviewed conference (see publications list)

Conference design for NSF's AdvanceVT project: Was part of six-member research group to design a conference for women entering academe in the sciences and engineering

- Conducted extensive literature review on learning theories, human cognition, gender bias, etc.
- Conducted interviews and focus groups with women and faculty members in academe to determine problems and potential solutions for increasing female representation in sciences and engineering
- Coined the term "Social Affordance" and created four design heuristics grounded in human information processing theory for use by future conference designers
- Work published in peer-reviewed journal (see publications list)

Navigation assistant on a PDA: Led a group of four graduate students in the design of a PDA-based application to facilitate independent and easy navigation for new visitors to Squires Student Center at Virginia Tech

- Analyzed the effectiveness of text vs. graphic modes for providing directions

Professional Memberships

- Member, **Alpha Epsilon Lambda (AEL)**, Academic Excellence and Leadership Honor Society for Graduate and Professional Students
- Member, **Upsilon Pi Epsilon (UPE),** National Computer Science Honor Society
- Member**, Center for Human-Computer Interaction,** Virginia Tech
- Member, **ACM Special Interest Group on Computer-Human Interaction** (ACM SIGCHI)