ABSTRACT

SRIKANTH, HEMA L. ADaPT: Adaptive Development and Prototyping Technique. (Under the direction of Ana I. Antón.)

Adaptive prototyping focuses on developing software for rapidly changing environments while improving delivery speed. Traditional methodologies are not effective in highly volatile environments; thus, agile methodologies have gained acceptance recently. Although agile methodologies offer less bureaucratic software processes, they fail to adequately support requirements engineering best practices. Adaptive prototyping provides a balance between heavy and ad hoc processes, aims to eliminate the drawbacks inherent in Agile Methodologies and traditional prototyping; it also incorporates requirements engineering best practices.

This thesis proposes a methodology, ADaPT (Adaptive Development and Prototyping Technique), which employs scenario analysis to elicit and validate requirements; maintains the spirit of CMM-level 2; iteratively re-examines system requirements; and only documents essential artifacts. Significant emphasis is placed on testing; and acceptance tests are written before implementation. Initial validation efforts, in the form of post-project surveys, suggest that ADaPT can improve system delivery speed and quality. Surveys were administered to three groups: customers, instructors, and students. Customers surveyed *agreed* that sponsored teams: delivered their system on time, developed a high quality system, and produced useful artifacts. Instructors surveyed *strongly agreed* that all projects were completed successfully, met course requirements and ensured a highly satisfied customer. The collective application of these techniques appears to improve software quality, reduce software cost, and improve system delivery speed while enforcing requirements engineering best practices as compared with previous experiences in student projects.

ADaPT: Adaptive Development and Prototyping Technique

by
Hema L. Srikanth

A thesis presented to the Academic Faculty of
North Carolina State University
in Partial Fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER SCIENCE
May 2002

APPROVED BY:

Dr. Ana (Annie) I. Antón,
Chair of Advisory Committee

Dr. Laurie Williams

Dr. Julia Earp

Biography

Hema Srikanth is originally from Bhilai - a small town in Madhya Pradesh, India. She completed part of her undergraduate course requirements in Economics at Pennsylvania State University, State College, PA and received her M.B.A from Indiana University of Pennsylvania. She worked in retail management for a few years before returning to the academic world. She was a postgraduate student in Computer Science at Clemson University, before joining North Carolina State University to pursue her Master of Science degree in Computer Science. She worked as a financial analyst at Advest, Inc. during Spring 1996; her project during the four month period resulted in several successful investment undertakings for the company. Hema likes to spend her spare time in volunteering activities- participating as a cook at IFC, and a mentor at Community Learning Partners. She plans to graduate from North Carolina State University in July 2002.

Acknowledgements

I am grateful to my parents for their sustained support. I am grateful to my husband, Manoj Nair, and for being so considerate and understanding during the past few months. I owe special thanks to my brother for his advice and support.

I would like to thank my committee members, Dr Antón, Dr Earp, and Dr Williams for their feedback during this past year. I am grateful to Dr Antón for her time and continued support; she has been an advisor and a mentor to me for the past two years. Special thanks to Dr Williams for taking time to read my thesis several times and her valuable comments led me to better my thesis. I wish to thank Dr Earp for her time and consideration.

I am grateful to Dr Rappa and Aldo Dagnino for partially funding my thesis. I would also like to thank Tony O' Driscoll for his time and help during the initial stages of my research.

Finally, I would like to thank Ryan Carter, Laura Tateosian, Kera Bell, Thomas Alspaugh, James Jong, William Stufflebeam for allowing me to vent every once in a while.

Table of Contents

List of Figures

Introduction

An idea is a feat of association. - Robert Frost

Traditional software processes, while rich with support for requirements activities, are not especially well suited for rapid software development. Agile software processes [Fow01a] have recently received increased attention due to the need to develop software for rapidly changing environments; however, we believe they fail to properly support essential requirements engineering practices. This thesis proposes an adaptive approach to software development, ADaPT (Adaptive Development and Prototyping Technique), which employs proven scenario and goal-based analysis techniques to elicit and structure requirements, ensuring that system requirements are iteratively examined via prototyping. ADaPT incorporates the concept of "user stories" in Extreme Programming [Wel01] and scenarios as traditionally used in requirements engineering. While introducing rigor into requirements activities, agility is maintained in ADaPT by documenting only the most essential elements. Validation is currently underway on several software development efforts that employ the model to support rapid development of electronic commerce applications.

1.1 Software Process Models

In today's fast-paced and competitive world of commercial software development, speed and flexibility are increasingly important. Companies are forced to try newer ways of developing software, moving away from traditional approaches including the waterfall and spiral models [Jal00], which are not as viable in today's rapid software development environment [CAD01]. Many projects are cancelled or fail to meet the customer's expectations. Studies have found that about two-thirds of all projects substantially overrun their estimates [SB01]. Since the early

1970’s, a large number of lifecycle models have been introduced. The most predominant one was the traditional “Waterfall” model [Roy90], which has been around for many years and works well on low-risk predictable projects. However, the waterfall model may not work very well for complex projects with changing requirements [Bea99]. The waterfall model has been observed to cause software to be more expensive, delivered later, and be more unreliable [DBC88].

In the 1980’s, Barry Boehm’s spiral model [Boe88] was introduced as a way of reducing risk. In the late 1980’s “Evolutionary” lifecycle models were introduced, where the goal is to “evolve” or “grow” some or all of system’s functionality into the final product iteratively [Gra89]. Evolutionary lifecycle models are iterative in nature. While the project plan for an entire project is developed in the beginning, it is revised at the end of every subsequent iteration [Hig98]. With the introduction of each new lifecycle, software developers have seen a slight decrease in the amount of required documentation. However, planning continues to require a substantial amount of resources (time and effort) throughout the life of a project, resulting in decreased delivery speed. Evolutionary approaches basically incorporate mini-waterfalls within each development cycle [Hig98]. Although traditional approaches might be effective for safely critical systems, they are not especially well suited for e-commerce system development [CAD01].

In the late 1990’s Agile Methodologies have gained popularity. Agile methodologies shift the focus away from project documentation to techniques used to develop/write source code. Although agile methodologies are less bureaucratic in nature than traditional software methodologies, they do not effectively incorporate requirements engineering best practices [Lei01]. As part of this research, we address the need for improved requirements practices in agile methodologies and the need for more flexibility and discipline in market-driven environments where software processes tend to be relatively ad-hoc. We propose ADaPT as a viable approach to rapid software development, which seeks a balance between too much and too little process. Adaptive lifecycles are believed to work well for systems with changing requirements.

E-commerce developers are faced with the challenge of developing systems with limited resources: cost, time and effort. The developers are constantly under pressure to develop systems at record pace, and the managers are under pressure to market the systems before competitors can release their systems. We introduce an adaptive prototyping technique, which is highly suitable for e-commerce systems.

1.2 Adaptive Development and Prototyping Technique

This thesis promotes adaptive prototyping as a viable approach for developing software in today’s rapid development environment. This section gives an overview of the Adaptive Development and Prototyping Technique (ADaPT), which is detailed in Chapter 3. Adaptive prototyping is believed to be a cost-effective way to develop software when the requirements are complex and evolving.

ADaPT enables software teams to produce software incrementally. Teams are more guided by their experience than by formally pre-defined project plans. ADaPT guides developers to work together effectively, enabling complex products to be developed efficiently. ADaPT has been initially validated within the electronic commerce and web-based application domains (discussed in Chapter four). Since the ADaPT lifecycle is component based, we believe, it is well suited for developing large complex systems. Our future work includes validation of this. ADaPT provides the following benefits:

- increased customer involvement to improve clarification of user requirements;
- improved ability to handle risk and uncertainty;
- better quality software; and
- increased development speed.

ADaPT is sensitive to changing requirements as evidenced by customer validation of the evolving system during every cycle. The requirements are revisited every cycle based upon customer validation and feedback.

ADaPT presents a risk-driven adaptive technique that explicitly addresses the risks and challenges inherent in small, rapid prototyping projects. For example, e-commerce software developers are often under pressure to develop software at record pace, often without software process guidance or models. This lack of process awareness makes it difficult for e-commerce developers to handle changing requirements effectively.

A team of five software engineers tailored the Software Engineering Institute's (SEI) CMM [PWC94, ACS01] to fit the needs of small development teams in e-commerce domains. We used this tailoring as our basis for ensuring that ADaPT adheres to the spirit of the Capability Maturity Model (CMM), as elaborated in Section 3.4.

ADaPT recommends minimal documentation while maintaining high software quality. The reduction of effort due to less documentation is believed to improve delivery speed. Most documentation is maintained in spreadsheets; the documentation pertaining to each component is completed during the cycle in which a given component is developed. The project plan is maintained at a very high level.

The better part of this thesis focuses on the detailed description of the risk-driven, CMM compliant, adaptive prototyping technique (ADaPT). The discussion focuses on the model's ability to overcome the weaknesses inherent in "traditional" prototyping models and agile methodologies.

The remainder of this thesis is organized as follows. Chapter 2 describes relevant work in the areas of agile methodologies, goal based software development, risk mitigation during system lifecycles, and CMM tailoring and use. ADaPT is introduced and discussed in detail in Chapter 3.

Chapter 4 summarizes the validation efforts made to verify the different facets of the model. Finally, Chapter 5 emphasizes ADaPT's research contributions and plans for future work.

Chapter 2 A Survey of Related Work

The next best thing to knowing something is knowing where to find it. – Samuel Johnson

---

This chapter provides an overview of the relevant related work. ADaPT incorporates some of the practices that are more prevalent in "*Agile Methodologies*". An in-depth study of the practices of these Agile Methodologies, conducted as part of this research is discussed in Section 2.1. Section 2.2 discusses goal-driven approaches for requirements engineering. Sections 2.3 and 2.4 highlight the core aspects of adaptive prototyping with respect to risk management during requirements engineering and the CMM (Capability Maturity Model), respectively. Section 2.5 discusses the Evolutionary Prototyping with Risk Analysis and Mitigation (EPRAM) model.

2.1 Agile Methodologies

Many software processes lack rigorous discipline, often characterized as "code and fix"[Fow01a]. The software is likely to be written without proper planning, and the design of the system may reflect short-term decisions. In ad hoc development, bugs are discovered increasingly in the latter stages of the lifecycle, and therefore the product is not released on schedule. An alternative way of developing software is to adopt a more disciplined approach, which ensures software efficiency and predictability. In disciplined approaches, significant emphasis is placed on project planning and the development process is documented and monitored in great detail. These methodologies are often bureaucratic and result in heavy overhead costs and reduced delivery speed [Hig98].

In response to the above methodologies, some practitioners have eagerly adopted a new class of methodologies called the "*Lightweight Methodologies*", also referred to as *"Agile Methodologies,"* which have come into emergence in the last few years [Hig01]. Agile Methodologies are best suited for smaller teams with fewer than ten individuals, and smaller projects. Agile methodologies are more suitable for smaller projects as focus shifts away from documentation and initial planning of the project, and towards techniques used to develop/write source code. Therefore, lesser documentation is believed to bring significant savings to cost and effort. The key characteristics of agile methodologies are summarized below.

- Agile methods are adaptive rather than predictive. Heavy methods (such as Waterfall) plan out a large part of the software process in great detail before beginning the project; this works well until things change. So waterfall models are resistant to change. In contrast, agile methods welcome change; they are adaptive processes that actually thrive on change [Fow01a].

- Agile methods are people-oriented rather than process-oriented. They explicitly make a point of working with peoples' nature rather than against them and to emphasize that software development should be an enjoyable activity [Fow01a].

ADaPT seeks to reduce excessive planning and documentation that is inherent in many evolutionary and traditional models, making it similar to agile methodologies in many ways. The following subsection summarizes various agile methodologies, examined while developing the ADaPT model, including Extreme Programming (XP) [Bec00], Scrum [Sch00, SB01], Crystal [Coc99] and ASD (Adaptive Software Development) [Fow01a]. The key practices of these analyzed methodologies are discussed below.

2.1.1 Extreme Programming

Extreme Programming **(**XP) is a disciplined approach to software development; it was developed to best suit projects with dynamic requirements. XP has proven to work best for smaller projects where user involvement is significant [Bec00]. XP begins with four values: Communication, Feedback, Simplicity, and Courage. XP programmers communicate with their customers and fellow programmers. They keep their design simple and clean. They deliver the system to the customers as early and often as possible and implement changes as suggested. Several other practices are followed along with these four values to successfully implement a project using XP, which enables XP programmers to respond to changing requirements, resources and technology [Bec00].

In XP, a project is broken down into several pieces and a fraction of the project is developed completely during every cycle. The requirements are gathered as user stories ("Brew some Coffee" is an example of a user story for a coffee maker) and one or more user stories are developed every cycle. The customer writes the user stories, which represent the system needs from the customer's standpoint. User stories focus on user needs and are very brief: usually three sentences long. After user stories are created, a release plan meeting is held. During the release plan meeting, a release plan is generated which details the estimated time needed to develop each user story and the customer's priority for each story. At the beginning of each iteration, an iteration-planning meeting is held and the customer chooses user stories for implementation during the next iteration. The iteration-planning meeting is also called a "planning game". Subsequent to the planning game, the stories selected for implementation are further elaborated into programming tasks, which must be accomplished for the successful completion of a user story. Acceptance tests are written by the customer for user stories; these tests are run during and after the iterations to ensure the user story has been implemented. A team of two is paired and implementation is performed in pairs. Coding standards are followed to keep the code consistent for the entire team to read and refactor.

XP puts testing at the foundation of development; unit test cases are written before the implementation of a given user story. The user story's implementation must pass all unit tests to be integrated into the system prototype. Unit tests are deposited into the code repository along with the tested code. Developers integrate and deposit code into the code repository frequently (every few hours). The customer develops acceptance tests; acceptance tests are intended to demonstrate that a customer's requirements are met by the system developed.

In XP, the customer provides ongoing feedback at the end of every cycle, which allows developers to change the functionality, accordingly, to fit customer needs and/or to improve user acceptance. The planning game is played at the end of every cycle, which allows the customer to reprioritize or make trade-offs for the next iteration.

2.1.2 Scrum

Scrum is ideal for small projects where the team size is four to seven individuals, but can also support large teams [Sch00, SB01]. It divides a project into several iterations. Development teams appoint a Scrum master at the beginning of the lifecycle. Scrum cycles are called *Sprints* and each sprint lasts no more than a month. The system is reduced into *backlogs* and each *backlog* is a set of features. The functionality to be developed in a sprint is defined and divided into tasks at the beginning of the sprint, and the development team is responsible for its delivery. The tasks are listed for a backlog and planning is based on the backlogs developed during a sprint [RJ00]. The product owner is responsible for creating the product backlog. Unlike traditional models, complete requirements for the project are not written upfront, as the customer is indecisive at the beginning of the project.

The key practice of Scrum is the *Scrum Meeting.* The development team holds a short fifteen-minute meeting everyday, during which the team discusses the deliverables for the following day. The team also discusses the *issues/blocks* that need resolution. At the end of each Sprint, there is a demo to:

- *show* the customer the product;
- *give* the developer's sense of accomplishment;
- *integrate* and test a reasonable portion of the software developed; and
- *ensure* real progress-reduction of backlog and not just papers [BDS00].

2.1.3 Crystal

Alistair Cockburn describes software development as a cooperative game where all team members' work towards achieving a common goal and where the end point is the running system software, code, and packaging. "Software development is a cooperative game, in which people use markers and props to inform, remind and inspire themselves and each other in getting to the next move in the game. The endpoint of the game is an operating software system; the residue of the game is a set of markers to inform and assist the players of the next game. The next game is the alteration or replacement of the system, or creation of a neighboring system" [Coc99]. He also emphasizes the need for different development styles for different projects as well as tailoring the existing methodology to meet the project's needs.

In Crystal, roles and tasks for a team vary depending on the size of the team. Crystal advocates rich communication between developers and customer. Projects are developed in small fractions with frequent delivery. Documentation overhead is minimal, but dependent on system size. System requirements are gathered while designers discuss system requirements with user/sponsor and write requirements as usage scenarios. Use of pair programming [WKC00] is recommended for implementation. Whiteboards are used to discuss design and these whiteboard prints are used to document the design. User screens are documented with pencil sketches. Release schedules are documented in short lists and customer involvement is essential.

2.1.4 Adaptive Software Development

Highsmith views planning as a paradox in an adaptive environment because outcomes are naturally unpredictable [Hig98]. In traditional lifecycles, deviations from plans are considered to be mistakes that should be corrected. However, in ASD (Adaptive Software Development) deviations guide developers to achieve the correct solution [Fow01a]. ASD developers must collaborate effectively to deal with risks and uncertainty. Creative ideas are generated through ongoing communication amongst team members; learning challenges stakeholders, developers and customers to examine their assumptions and to use the results of each development cycle to adapt to the next [Fow01a].

The ASD lifecycle has six main characteristics [Hig00]; ASD is:

- Mission Focused-Overall mission of the project is well documented to measure progress.
- Component Based-Components are a group of features developed during a cycle.
- Iterative-Components evolve over cycles as customers provide feedback.

- Timeboxed-The progress of the project and its mission should be constantly evaluated. Timeboxing determines the time required for the project, and measures the progress made.
- Risk Driven-Component cycle plan is initiated by analyzing risks.
- Change Tolerant-ASD has the ability to incorporate changes into the product.

In ASD, the traditional and static Plan-Design-Build approach is replaced with a Speculate-Collaborate-Learn approach. We now discuss the concepts of Speculate-Collaborate-Learn as follows [Hig00]:

Speculate in ASD has seven steps:

- perform project initiation phase;
- determine the project timebox;
- determine the total number of required cycles and timebox for each;
- define an objective statement for each cycle;
- prioritize the components and assign them to cycles;
- assign technology and support components to cycles; and
- develop a project task list.

Collaborate in ASD involves-Concurrent Component Engineering: During this phase, the identified components are delivered. The phase begins by assigning the components to the development team, and allowing the team to deliver the components without any management supervision.

Learn in ASD involves four categories of elements to learn and evaluate at the end of each development cycle [Hig00]:

- the product component's quality from the customer's perspective;
- the product component's quality from the technical perspective;
- the performance of the delivery team and the practices that were used; and
- the project status.

There is another form of review, performed after every component cycle, which is fed into the planning efforts of the subsequent cycle. This review analyzes the project's progress and how it measures up to its plans. The review also draws a comparison of where the project should be at the end of current cycle and where the project currently is.

2.2 Goal Driven Approach to Requirements Engineering

Software projects need to be developed based on a planning, execution and feedback model derived from past experience. Sound planning involves setting of project goals, which includes defining the system that is being built. Sound execution includes *identifying* the scenarios (descriptions of events and sequential behaviors for an existing or desired system) that assist in understanding the functionality of the system and *operationalizing* the requirements of the system using identified scenarios. Sound feedback includes capturing the experience gained from the current project for use in future projects. This section discusses the goal-based refinement of requirements [Ant96, Ant97], which is used to show an operational example of ADaPT.

A critical factor in a project's success is for developers to not only understand *what* they are developing, but *why* they are developing a given system [Ant97]. Goals are objectives for accomplishment, which provide a framework for a desired system. Goal-driven Requirements Engineering (RE) approaches focus on why systems are constructed, providing the motivation and rationale to justify software requirements [Ant97]. Goals are a logical mechanism for identifying and organizing software requirements. The use of goal hierarchies to explore and represent the relationships between goals and scenarios are documented in the literature [AP98, DvLF93] and employed by ADaPT. It is easy to overlook and difficult to uncover requirements using traditional RE techniques [PTA94]. Goals (the targets of achievement) and scenarios (behavioral descriptions of a system) [AP98, DvLF93, Lam01, RSB98, vLDM95, WPJ98] have proven to ensure the early identification of typically overlooked requirements [PTA94, AP98].

The GBRAM (Goal Based Requirements Analysis Method) supports goal-based requirements engineering via the provision of strategies for the initial identification and construction of goals. The method includes a set of guidelines and recurring question types that suggest goal identification and refinement strategies and techniques.

The GBRAM (Goal Based Requirements Analysis Method) employs a goal hierarchy to structure and organize requirements information (i.e., scenarios, constraints and auxiliary notes, such as rationale) [Ant97]. The goal hierarchy aids analysts in finding information and sorting goals into naturally different functional requirements. Heuristics are useful for identifying and analyzing specified goals and scenarios as well as for refining these goals and scenarios. The GBRAM heuristics and supporting inquiry include references to appropriate scenario construction [PTA94, AP98] and the process by which they should be discussed and analyzed. The ADaPT planning and design phase employs the GBRAM to support the identification, elaboration and refinement of scenarios (during initial requirements gathering) and goals for requirements operationalization. ADaPT uses this technique of goal refinement on the identified scenarios (during initial requirements gathering) to operationalize requirements.

2.3 Managing Risk During Requirements Engineering

Risk analysis during adaptive software development is a key factor for project success. In ASD, planning involved for subsequent cycles are driven by risk analysis [Hig00]. Boehm's Spiral and Win-Win models rely upon explicit risk analysis techniques [Som95, BI96] to manage uncertainty and risk. However, managing risks during the software lifecycle is challenging in rapid development environments [CAD01].

Many industries use decision-making business models to manage risk during the development phases of a product [Car01]. For example, ABB (Asea Brown Boveri) uses a phased business decision-making model, referred to as Gate Model, to minimize the risk that a development cycle will spin out of control [CAD01]. The model is believed to deliver a high-quality product to the customer.

ADaPT is risk-driven, where risks are identified, analyzed and managed before and during every cycle. It relies upon weekly risk-analysis meetings to formulate a comprehensive risk mitigation strategy. The project risks are evaluated at the beginning of the project and subsequently at every risk-driven meeting. ADaPT supports the compliance of the documented system requirements with established security and privacy policy [AE01a] via risk analysis meetings.

2.4 Capability Maturity Model (CMM)

Many software organizations are striving to improve their software process by utilizing the process improvement structure outlined in the CMM. However, smaller organizations, due to lack of resources, are experiencing difficulties in adopting the CMM [OC99]. Customization is often required to support the varying needs of different organizations and tailoring the CMM to adapt to organizational needs is an accepted practice [Pau98, Pau99].

We examined a case study of Winapp, a company of five employees that develops PC based client server software applications. Winapp tried to improve its software process maturity [OC99] as the company was facing increasing difficulty tracking project's status. The company was unable to handle large and complex projects effectively and therefore decided to explore CMM to make software process improvements. The changes mandated by CMM (planning and required documentation) were hard to implement due to lack of resources (time and manpower). The company decided to determine and apply the most essential concepts of CMM process improvement and employed a better framework with selective key process areas [ACS01, Car01]. Although the company incurred additional overhead, the creation of this new framework led to significant improvements; developers perceived a 157% increase in requirements analysis activities and a 57% decrease in the number of requirements faults [OC99]. These figures, although subjective, provide sufficient reasons to improve software process within small organizations. A team of five software engineers, at North Carolina State University, tailored the CMM, to more appropriately support the demands of small software development teams working with limited resources. This tailoring serves as the basis for ensuring the ADaPT adheres to the spirit of the CMM [ACS01]. The report detailing the tailored CMM [ACS01] is included in references.

2.5 EPRAM

Researchers in North Carolina State University have been addressing the need for requirements rigor during rapid software development [ACE01, ACS01]. The requirements engineering team at NCSU developed EPRAM and employed EPRAM on nine web-based e-commerce development efforts.

EPRAM is a risk-based software development methodology, which addresses the challenges prevalent in small, rapid development efforts [CAD01]. EPRAM extends traditional evolutionary models [Dav92] with an added risk management factor as originally identified by practitioners in [BS96]. EPRAM is a CMM-compliant (Level 2) software process model, which employs a risk mitigation strategy to minimize the ill effects of requirements creep by advocating early detection and resolution of requirements conflicts. EPRAM was validated on several e-commerce projects where security and privacy were vital.

Four cycles comprise the EPRAM model. The project plan is created during the first cycle and then maintained throughout the project life. System requirements are reevaluated for consistency during each prototyping cycle. The business plan and requirements document are also created during the first cycle. The design document is created during the second cycle. A system prototype is created in the third cycle and implementation and testing is performed in the fourth cycle. Risk mitigation meetings are held at the end of every cycle. All documentation is updated during every cycle to reflect all updates. Quality is enforced in the designed system by ensuring compliance to CMM and conducting risk analysis during every cycle. ADaPT was designed to address the weaknesses of the EPRAM model as described in Chapter 3.

2.6 Summary

This chapter presented an overview of various Agile Methodologies and their respective practices. Additionally, it addressed topics that summarize the background research conducted as it pertains to adaptive prototyping, risk mitigation, and the CMM tailoring. Chapter 3 presents ADaPT, which is designed to improve software quality and software delivery speed. The technique offers the benefits inherent in evolutionary and agile prototyping models while avoiding their drawbacks.

Chapter 3 Adaptive Development and Prototyping Technique

For a successful software engineering methodology, pragmatics must take precedence over elegance, for Nature cannot be impressed. - Coggins' Law of Pragmatic Software Engineering

This thesis proposes an adaptive approach to software development, ADaPT (Adaptive Development and Prototyping Technique), which employs proven scenario and goal-based analysis techniques to elicit and structure requirements, ensuring that system requirements are iteratively examined via prototyping. The ADaPT methodology was designed to improve the software quality and delivery speed of rapidly developed systems. The methodology helps manage changing requirements, yielding an adaptive software process. Traditional software processes models, while rich with support for requirements activities, are not especially well suited for rapid software development [CAD01]. ADaPT incorporates the concept of "user stories" from Extreme Programming and scenarios, as traditionally used in requirements engineering. While introducing rigor into requirements activities, agility is maintained in ADaPT by documenting only the most essential elements. Preliminary validation has been completed on several software development efforts that employ the model to support rapid development of electronic commerce applications.

An overview of ADaPT and its phases are discussed herein. ADaPT's planning and design phase as well as implementation and testing phase are described in Subsections 3.1.1 and 3.1.2, respectively. Section 3.2 provides a mini tutorial for applying ADaPT.

3.1 ADaPT

Two main phases comprise ADaPT: (1) the planning and design phase and (2) the implementation and testing phase. During planning and design, developers elicit requirements in the form of scenarios and goals, and begin project planning. During implementation and testing, developers employ pair programming to write the code and conduct extensive testing. Figure 3.1 provides a high level overview of ADaPT. The ovals portray high-level project team activities and block lines represent quality assurance activities. ADaPT calls for basic essential documentation to ensure an adaptive process. Project planning and requirements documentation are maintained in spreadsheets; each subsystem's documentation is written during the cycle in which it is developed. Software quality assurance and configuration management documents are written concisely at the beginning of the project lifecycle. As previously mentioned, risks are evaluated during risk analysis meetings (as shown in Figure 3.1).

A more detailed overview of the ADaPT process is provided in Figure 3.2. The oval-shaped figures represent process activities, curved-rectangles represent the documentation artifacts, thicker arrows represent major control flows through the process, and the narrower arrows indicate data flowing as a result of the activities. The squares represent the processes that take place throughout the cycle. We now discuss each of these aspects of the model.

3.1.1 The ADaPT Planning and Design Phase

During the ADaPT planning and design phase, developers elicit requirements for a project and begin project planning. Two artifacts are produced during this phase: the *planning workbook* and the *requirements workbook*. The planning and design phase activities are limited to three weeks of time.

An initial meeting between the customer/user and the development team is held to gather information about the desired system. The ADaPT methodology has been designed to best suit a development team of twelve or less individuals and a system testing team of two or less individuals. A customer is the individual or entity for whom the system is designed; and who participates in meetings with the development team. During this initial meeting, the developers create scenarios that reflect the system as described by the customer/user. A scenario is a behavioral description of a system [PTA94]. The collected scenarios are analyzed and listed as goals that have to be achieved for successful development of the system. A goal is a target of achievement, which provides a framework for a given system [Ant97]. The goals are refined into *operational requirements* for the system. A requirement specifies how a goal should be achieved by the system. Similarly, ADaPT employs goal analysis to drive the planning process.

Figure 3.1: High Level Overview of ADaPT

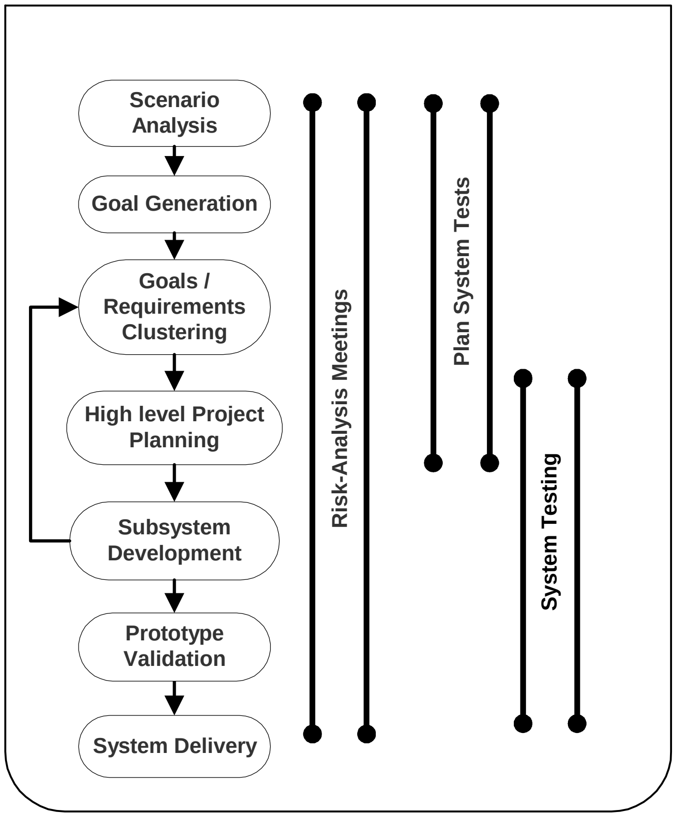


Figure 3.1: High Level Overview of ADaPT

Figure 3.2: ADaPT

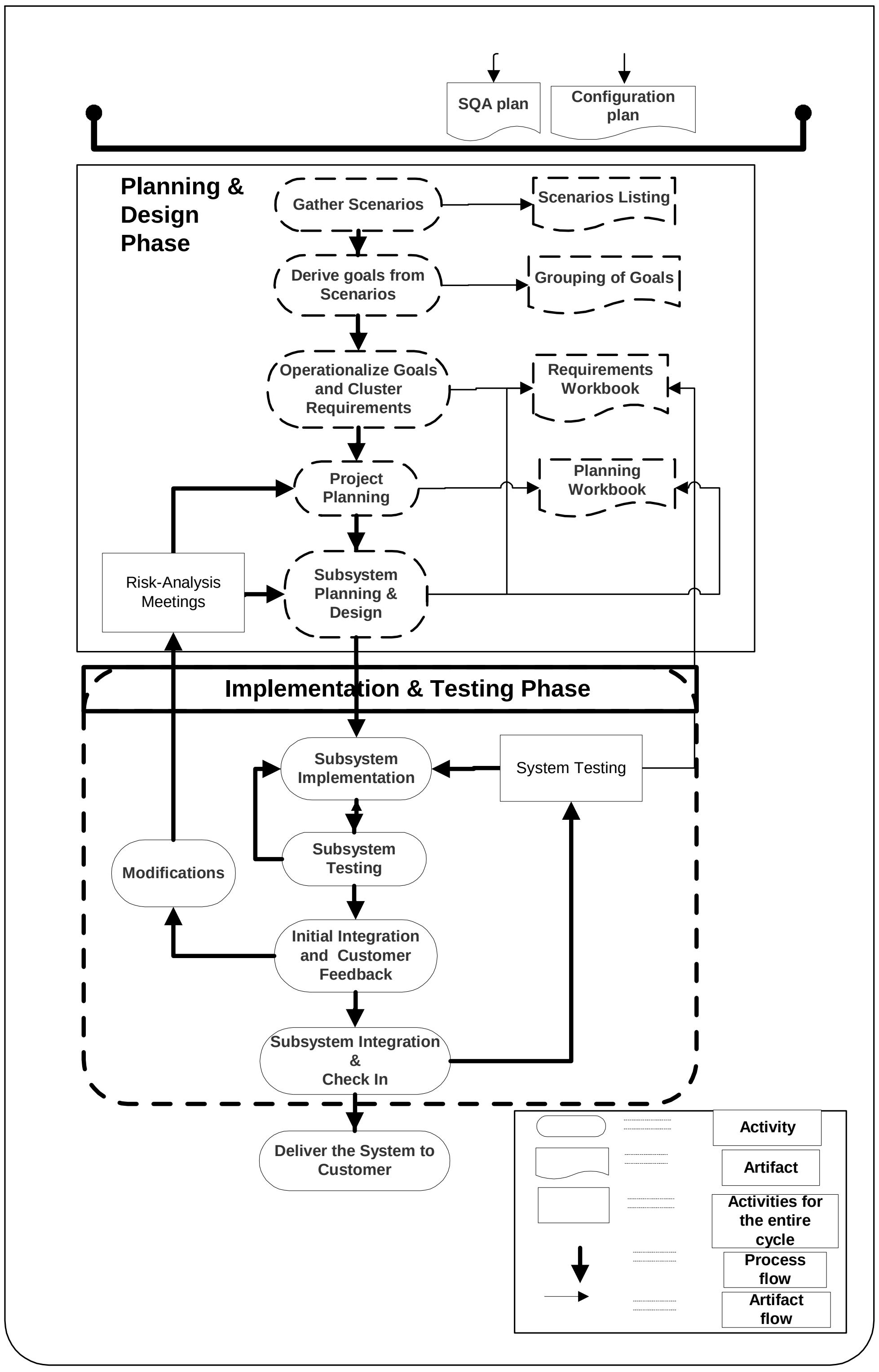


Figure 3.2: ADaPT

The planning phase is comprised of the following five main activities.

Activity 1: Scenario Analysis

During the initial meeting between the customer and the development team, the customer and other stakeholders describe the system needs to the development team. During this and subsequent meetings with stakeholders, scenarios that reflect the system as described by the customer/user are created. These scenarios are documented in a text document; as few as five scenarios or as many as several hundred scenarios may be documented. Consider the following six scenarios for an online shopping site:

S_1: Search for products

S_2: Check product availability

S_3: Compare product features/price

S_4: Add item to the shopping cart

S_5: Register

S_6: Complete purchase

Scenarios such as these would serve as the basis for goal elaboration and subsystem design, as we now discuss. For the remainder of this section we employ this online shopping site example to describe the ADaPT process activities.

Activity 2: Generate Goals

The collected scenarios are analyzed and later elaborated with goals that must be achieved. In ADaPT these goals ultimately represent operational requirements. Analysts may employ various techniques to analyze the scenarios such as the Inquiry Cycle Model [PTA94] or the GBRAM [AP98]. Using the GBRAM we elaborate two of the above scenarios with the goals required to satisfy each scenario as follows.

S_1: Search for products

G_1: (System) PROMPT user to enter search keywords

G_2: (System) SEARCH for keyword matches

G_3: (System) GENERATE search results web page

G_4: (System) DISPLAY search results web page

S_3: Compare product features/price

G_1: (User) SELECT products to be compared

G_2: (System) SEARCH for product features

G_3: (System) GENERATE table with product comparison info

G_4: (System) DISPLAY search results web page

Note that each goal represents an event comprised of an actor/action tuple, as in [AAB99]. Once the scenarios have been elaborated with goals, the goals are clustered according to Activity 3, below.

Activity 3: Cluster Requirements

The goals generated for each scenario are organized so that related goals form logical subsystems. Thus, subsystems are formed by clustering related goals; the approach is similar to the hierarchical approaches taken in [Ant97] and [DvLF93]. The requirements are clustered and documented in the requirements workbook (see Appendix B). A subsystem is a fraction of the system and the group of requirements that comprise a subsystem are implemented during a given subsystem development cycle.

The goals that form a subsystem may not necessarily come from one particular scenario. In other words, a subsystem can be comprised of some goals generated from S_1, some from S_3, and so on. In our example, goals G_1, G_2 and G_3 from S_1 may be grouped with goals G_1 and G_2 from S_3 to form a subsystem. All five goals address different kinds of searches and are thus clustered to form one subsystem, documented as a "search" subsystem in the requirements workbook. The documented subsystem and its respective requirements are revisited during the subsystem's implementation to ensure consistency and understandability. Only the requirements corresponding to the subsystem developed during a given cycle are listed in the requirements workbook and updated during that cycle.

Because the requirements that are part of a subsystem may not necessarily come from analyzing one particular scenario, one system may have several subsystems. For example: when developing a shopping center, the User Interface (UI) and the requirements generated for the UI may be grouped to form a subsystem. Similarly, the database (DB) and the requirements listed for the DB may be grouped to form a different subsystem. Changing or new requirements may then be incorporated into these subsystems since goal and scenario analysis inherently supports requirements evolution. For XP practitioners, an ADaPT scenario is roughly equivalent to an XP “user story” [Bec00].

Activity 4: Plan Project

The project planning activity is characterized as high-level and encompasses estimating time and resources needed to develop the product; this information is documented in the planning workbook (see Appendix B). We assume that the customer will employ the system designed by the development team using ADaPT; therefore the designed system is not a throw away prototype.

The project manager is appointed and the technical leaders responsible for overseeing each subsystem are appointed as well. The project manager is the individual who monitors the entire project, and the technical lead overlooks assigned subsystems. The project manager may be appointed before the project begins, however, the technical lead for the subsystems should be appointed after the subsystems have been clearly defined, since the assignment of technical leads to individual subsystems would depend on the expertise of the technical leads. The project plan includes: a brief system overview; a list of all team members and their respective roles; the prioritized list of subsystems; and the scheduled delivery date for each subsystem and the system.

Before the beginning of each cycle, which may last anywhere from 2 to 4 weeks, requirements are reevaluated for completeness and consistency; existing resources are evaluated to ensure fulfillment of the requirements; and requirements are checked for compliance with all security and privacy policies. The requirements planned to be incorporated into the system during a given subsystem development cycle may originate from several sources including, but not restricted to, policies (e.g. privacy and security policy requirements [AE01]), stakeholders (customers, users, and developers), and other projects. Requirements are agreed upon during a negotiation process that occurs between the stakeholders. The development team agrees to incorporate requirements into the system based on the organization's resource availability (time and manpower); these agreed upon requirements are incorporated into the system. These system-requirements are documented in the requirements workbook and are reflected in the system at the next customer evaluation meeting.

Once logical requirements are organized into subsystems, the subsystems are prioritized based upon a subsystem's dependency upon other subsystems; and the subsystem with the highest priority is developed first. During each cycle, subsystem planning is performed before its implementation, though one or more subsystems may be developed at a given time. The planning workbook contains a sheet for describing the "system overview" (see Appendix B); each subsystem is planned and documented in a different individual sheet as part of the same workbook. Subsystem design is discussed below.

Activity 5: Design Subsystem

One or more subsystems are chosen for development during a given cycle (see Figure 3.2). The subsystem requirements are re-evaluated before development begins. Each subsystem is broken down into several tasks; the tasks are then documented in the planning workbook. These tasks are assigned to team members and the scheduled completion date for each task and the subsystem are documented accordingly. The design issues pertaining to a subsystem are discussed using electronic whiteboards and documented in free form in the planning workbook. Copies of whiteboard drawings are maintained within the planning workbook. Design meetings focus solely on those subsystems being developed during a given cycle. Use-case diagrams may be used to show the design elements and inter-subsystem relationships.

Figure 3.3 graphically portrays the five planning phase activities. In this Figure, “M” represents the mechanism involved in the process activity, “A” represents the process activity itself and “D” represents the artifact or documentation as a result of the process activity.

Figure 3.3: ADaPT Planning and Design Phase

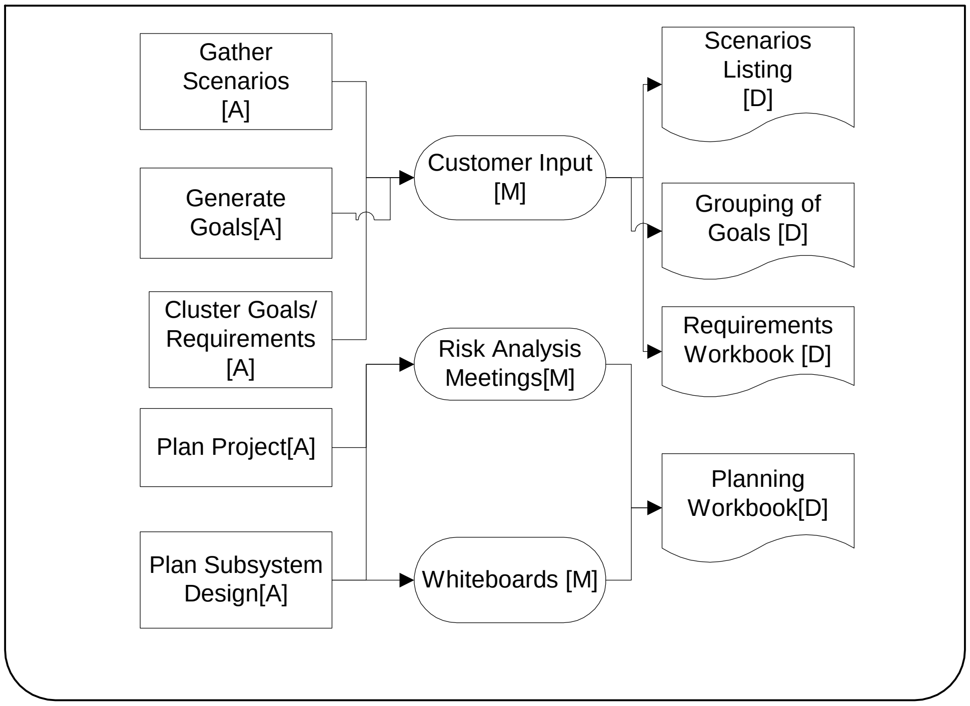


Figure 3.3 ADaPT Planning and Design Phase

3.1.2 The ADaPT Implementation and Testing phase

The ADaPT planning and design phase is followed by the implementation and testing Phase. This section describes the ADaPT implementation and testing phase. Every subsystem is developed according to Figure 3.2. Figure 3.4 graphically depicts the four key activities that comprise the subsystem implementation and testing phase.

Activity 1: Subsystem Implementation

Subsystem development cycles are fairly short and usually do not extend to more than a month. If a particular subsystem is estimated to take longer, it should be broken down into smaller subsystems.

The goal is to focus on each small piece (subsystem) one at a time and to do so thoroughly with proper planning and feedback from the customer.

Pair programming [CW00, WKC00] has worked well in situations where the requirements change frequently and the projects are complex. In pair programming two developers work together, as they take turns in writing code. One developer observes the other developer writing code; but the resulting source code reflects both developers' ideas. ADaPT employs pair-programming during implementation as a technique to improve software quality.

Activity 2: Subsystem and System Testing

Another way in which quality is addressed in ADaPT is via subsystem and system testing. Subsystem testing entails white box and black box testing. System testing is performed throughout the lifecycle to ensure all system elements are properly integrated. The system testing team is comprised of one or two individuals. A successful implementation of a subsystem includes passing all test cases for the subsystem. At the end of each cycle, every subsystem is tested thoroughly and integrated with the other subsystems. The elements in subsystem and system testing are described below.

Subsystem Testing: This is comprised of white box and black box testing:

- White Box Testing: **-** White box testing ensures that all program statements are executed, according to program structure and that expected results are achieved. After each subsystem is implemented, the developers responsible for the subsystem development test the structure and syntax of the code written to ensure its operation.
- Black Box Testing – After White Box Testing, each subsystem will be subjected to two tests described below: functional testing and acceptance testing. Black box testing focuses on program test cases that are based on the system specification, from the developers (functional tests) and customer's (acceptance tests) point of view.

1. Functional Testing – The subsystem tests are performed to verify that the subsystem meets the technical requirements as identified by the development team. The test cases will be created prior to the subsystem development and are documented in the requirements workbook.
2. Acceptance testing: - The subsystem is subjected to further tests to ensure conformity to customer requirements. Data will be input to the subsystem and the output will be verified to ensure conformance of the system with the user requirements. *Acceptance tests* for the subsystem may be written before the implementation of the subsystem, based upon the initially collected and documented scenarios. Acceptance tests are tests conducted to enable the customer to validate that the requirements for each scenario have been satisfied. The acceptance test to be conducted will be documented in the Requirements Workbook under the heading "Acceptance Test [AT] for each subsystem" in the respective subsystem worksheet.

System Testing

- Integration testing: After completion of subsystem testing during a cycle, the subsystem developed will be integrated with the system developed thus far. A subsystem has to pass all subsystem tests before being integrated with the system. This phase of testing will test the functionality of the newly integrated system.

Activity 3: Initial Integration and Customer Validation

The developers integrate the initial version of the subsystem developed in the current cycle with the system thus far. The customer evaluates the system prototype, along with the feature added during the subsystem development cycle (via acceptance tests) to ensure their software requirements are met. Any customer-suggested modifications (such as new or missing requirements) are addressed at this time. The requirements workbook is then updated with these requirements changes. The customers' validation at the end of every prototyping cycle enables developers to iteratively revisit the requirements and ensures risk minimization**.** Since the customer's opinion is taken before, during and at the end of every subsystem cycle, requirements changes during validation of the final cycle are expected to be minimal.

Activity 4: Final Subsystem Integration

After the suggested modifications are incorporated into the validated system during Activity 3, the revised subsystem is further integrated with the entire system. The system testing team is responsible for integrating each subsystem with the overall system. The subsystems are integrated with other subsystems and deposited (checked-in) in a repository after the modifications suggested by the customer are completed and customer satisfaction is ensured. Figure 3.4 graphically depicts the activities for the design and implementation phase of ADaPT:

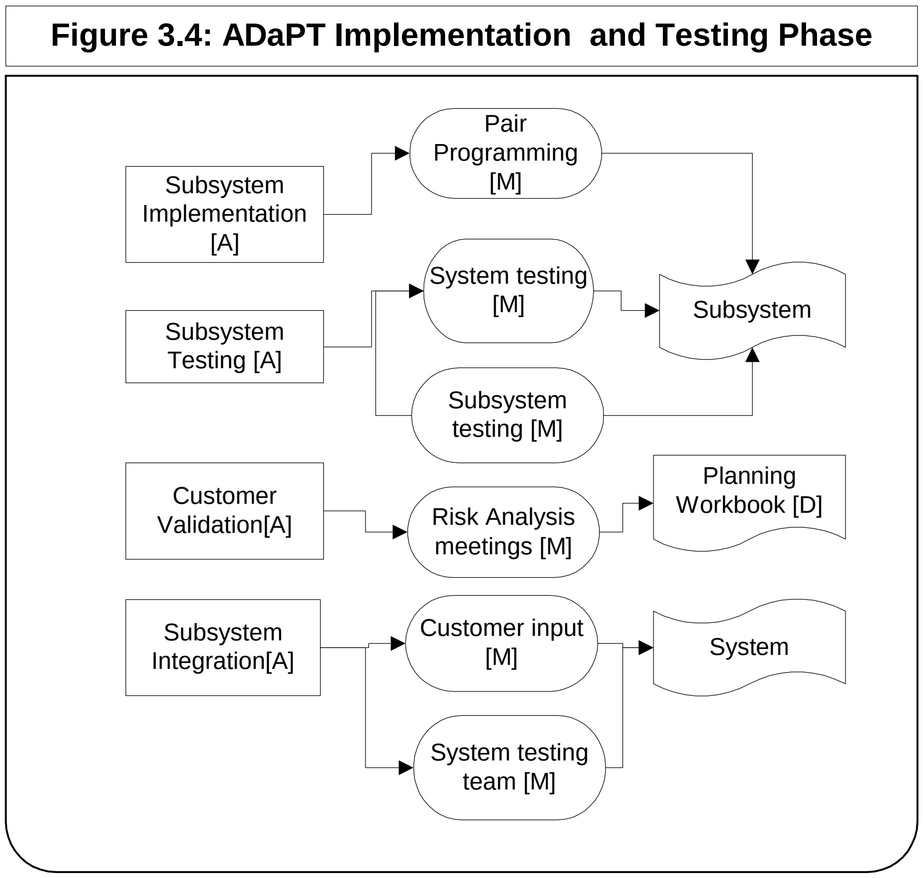


Figure 3.4 ADaPT Implementation and Testing Phase

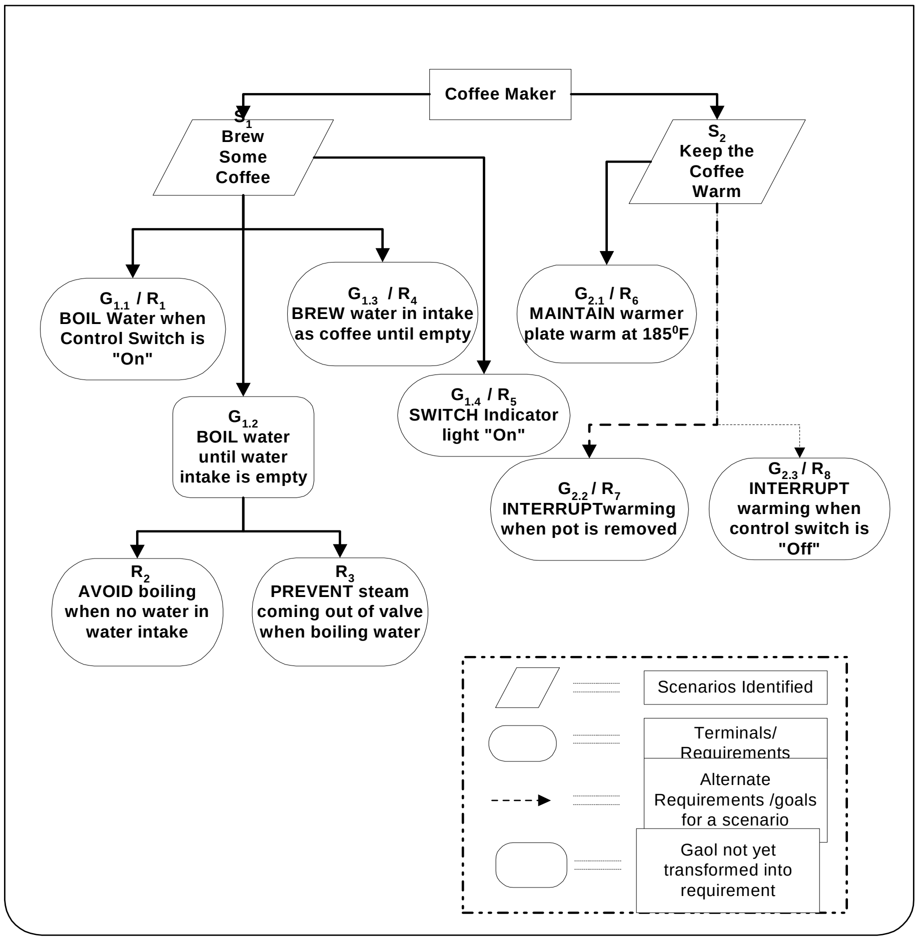
3.2 Operational Example of ADaPT

In this section, we demonstrate the application of ADaPT within the context of an ordinary 20-cup coffee maker system. The average cost of this type of coffee maker is twenty dollars; thus its functionality is rather simple. For convenience and ease of understanding, we elaborate only the system viewpoints while designing this coffeemaker.

In the example shown below, with the objective to design a simple coffee maker, two scenarios S_1 and S_2 are identified to elaborate the system viewpoint. We elaborated each scenario by identifying goals that satisfy the objective of each scenario by asking: *"What must the coffee maker do to satisfy this scenario?"* Every goal is further analyzed to determine if it may be further decomposed; ultimately those goals, which cannot be further elaborated, represent operational requirements. For example, in S_1 (Brew some coffee) the goal $G_{1.2}$ is further decomposed into two additional goals, namely R_2 and R_3. While analyzing S_2, we identified two alternative "interrupt" goals (shown in Figure 3.5 using dotted lines); the system should satisfy the requirement to interrupt warming when the "control switch is turned off" or "when the coffee pot is removed from the warmer plate."

Figure 3.5 graphically depicts the application of ADaPT while designing a coffee maker. The terminal nodes are the requirements identified for the system (shown as R_x in Figure 3.5).

Figure 3.5: Coffee Maker (System Viewpoint)



Coffee Maker
S1 Brew Some Coffee
S2 Keep the Coffee Warm
G1.1 / R1 BOIL Water when Control Switch is "On"
G1.2 BOIL water until water intake is empty
G1.3 / R4 BREW water in intake as coffee until empty
G1.4 / R5 SWITCH Indicator light "On"
G2.1 / R6 MAINTAIN warmer plate warm at 185⁰F
G2.2 / R7 INTERRUPTwarming when pot is removed
G2.3 / R8 INTERRUPT warming when control switch is "Off"
R2 AVOID boiling when no water in water intake
R3 PREVENT steam coming out of valve when boiling water
Scenarios Identified
Terminals/ Requirements
Alternate Requirements /goals for a scenario
Gaol not yet transformed into requirement

Figure 3.5 Coffee Maker (System Viewpoint)

S_1: Brew some coffee

$G_{1.1}$ / R_1: BOIL water when control switch is “On”.

$G_{1.2}$: BOIL water until water intake is empty.

R_2: AVOID boiling when there is no water.

R_3: PREVENT steam coming out of valve when boiling water.

$G_{1.3}$ / R_4: BREW water in water intake as coffee until water intake is empty.

$G_{1.4}$ / R_5: SWITCH indicator light “On”.

S_2: Keep the coffee warm

$G_{2.1}$ / R_6: MAINTAIN warmer plate warm at 185^0F temperature.

$G_{2.2}$ / R_7: INTERRUPT warming when pot is removed.

$G_{2.3}$ / R_8: INTERRUPT warming when control switch is “Off”.

After the requirements are generated, they are grouped into subsystems to be developed in a subsystem cycle. The requirements for manufacturing the coffee maker can be grouped into three subsystems.

SS_1: BOIL Water functionality: *R_1, R_2, R_3*

SS_2: BREW button Functionality: *R_4, R_5*.

SS_3: WARMER PLATE Functionality: *R_6, R_7, R_8*.

The two scenarios were elaborated with goals; these goals were allocated to three subsystems. These scenarios are similar to “user stories in XP [Bec00]; in XP these scenarios would require two development cycles. *User stories* are written to describe system needs by the customer [Bec00]. However, by grouping related requirements to form subsystems in ADaPT, developers might avoid redundancy and the implementation process will be more efficient.

3.3 Risk Mitigation in ADaPT

ADaPT addresses project related risks via weekly risk-analysis meetings. The development team and testing team hold weekly reviews to discuss project risks and maintain progress reports. “Requirements creep” refers to substantial modifications to the initially documented requirements for a software system, resulting in extensive alteration of the system’s existing functionality and scope [Car01]. In ADaPT, as in EPRAM [Car01], requirements creep and project risks are addressed during risk analysis meetings. The team members also use the risk analysis meetings to discuss design, address design constraints, identify subsystem dependencies, elaborate tasks for each subsystem and plan subsystem development.

In ADaPT, the customer is actively involved in providing feedback at the end of every development cycle, as the customer is actively involved with the development team in identifying system requirements before and during each cycle. The customer also validates the system after the successful implementation of each subsystem. Thus, at the end of the subsystem development cycle, the modifications, if any, are made to the system developed thus far. At the end of every cycle the customer's expectations are evaluated with the system at hand to ensure that development is progressing according to the scenario prioritization initially provided by the customers. ADaPT manages requirements changes effectively by minimizing requirements creep due to the focus on the scenario prioritization. Requirements creep is further minimized by involving the customer early on, providing frequent progress reports to the customer and via customer acceptance tests at the end of every cycle. The subsystem developed during a cycle does not go through final integration until the customer validation and any modifications are addressed.

3.4 Application of Tailored CMM to ADaPT

Researchers at North Carolina State University have been addressing the need for requirements rigor during rapid software development [ACS01, ACE01]. Their earlier work resulted in the development of EPRAM model [CAD01], which is discussed in Section 2.5. While developing EPRAM, a team of five software engineers participated in tailoring CMM-level 2 (Repeatable level) for introducing process in rapid development projects. The in-depth study of tailored CMM-level 2 is available as a reference [ACS01].

To tailor the CMM, five Key Process Areas (KPA) were examined in the Repeatable level (Level 2): Requirements Management, Software Project Planning, Software Project Tracking and Oversight, Software Quality Assurance, and Software Configuration Management. Sixth KPA: Subcontract

Management, was omitted as it was assumed to be irrelevant for smaller rapidly developing projects. The tailored CMM-Level 2 KPA’s are incorporated in ADaPT to uphold the spirit of CMM.

The tailored CMM was comprised of 16 goals, 6 commitments, 22 abilities, 49 activities, 5 measurements, and 16 verifications. We analyzed all of these elements for adherence to ADaPT and eliminated 7 verifications and 2 activities (as detailed in Appendix D). The verifications that required senior management process reviews were considered nonessential since the validation of ADaPT was performed on projects with team size of less than ten individuals, which had no senior management. The project team is comprised of one project manager, few technical leads, and few programmers. Additionally two developers form a system testing team to perform testing. The technical leads and programmers also fulfill additional roles in Software Quality Assurance and Software Configuration Management teams. Training for team members on software engineering activities entailed a combination of in-class, self-study and peer study activities.

ADaPT’s adherence to CMM is based on its compliance with the above mentioned elements. ADaPT differs from EPRAM in two ways: all documentation is maintained in spreadsheets; and documentation of each subsystem is performed only at the beginning of the subsystem development cycle with minor updates performed periodically.

3.5 Summary

In this chapter, we introduce ADaPT for rapid system development. We believe ADaPT improves software delivery speed by eliminating excessive documentation and improving efficiency by maintaining all documentation in the form of workbooks. Quality is addressed via testing in ADaPT. System requirements are clarified via the application of scenario and goal analysis to operationalize requirements (common best practices in requirements engineering). An operational example is provided to elucidate a simple application of the ADaPT model. In Chapter 4, we summarize our experiences to date with ADaPT and discuss our current validation efforts.

Chapter 4 Validation

Beware of the man who won't be bothered with details. - William Feather

---

ADaPT was applied in ten electronic commerce (e-commerce) and three web-based application development efforts to validate its usefulness. In this chapter we describe our validation efforts, which involved the use of ADaPT to develop software applications and surveys. Surveys were administered to students at the end of the project, as well as to instructors and customers to gauge their satisfaction with the process and the resulting product artifacts.

ADaPT was applied in two educational settings at North Carolina State University (NCSU): a graduate level e-commerce practicum course (CSC 591O/BUS 516) offered as a joint venture between the Computer Science department and the College of Management, and the undergraduate software engineering (SE) course (CSC326) in the Computer Science department. The practicum had twenty registered students divided into four multi-disciplinary teams of five individuals in each team. Each team developed e-commerce applications that were sponsored by five well-established and highly reputable institutions: IBM, North Carolina State University, Hickory Museum of Art, and Lipsinc and Centra (see Appendix D for project and team descriptions). The SE course had sixty-nine students divided into nine teams that developed projects sponsored by NCSU (see Appendix D for project and team descriptions).

The teams in both the practicum and the SE class used ADaPT as their software development lifecycle process to develop their systems over the course of 15 weeks during the Spring semester of 2002. The course instructors and the process liaison (the author of this thesis) allocated three weeks for *planning phase* activities and the remaining time for subsystem development and testing activities.

The students in the e-commerce practicum received training in how to apply and use ADaPT. They were also provided with the ADaPT Process Description guide (see Appendix A) and all associated templates (see Appendix B). The practicum students met weekly with the instructors and contacted the process liaison frequently to discuss any process-related concerns. The process liaison and instructors spent several hours each week reviewing the student-produced documentation to provide feedback to the students. During their weekly meetings, the students received additional informal training in ADaPT from the instructors and the process liaison.

In the senior level SE course, students received the same training, documentation and templates as the e-commerce students during lecture at the beginning of the course. Whereas weekly meetings with the instructor were mandatory in the practicum, they were optional for the SE students; this is due, in great part, to the different structure for each course. In contrast to the practicum course, in which students have very few lectures and no exams because the course focuses solely on application development, the SE course is a core course in the computer science curriculum, requiring students to complete homework assignments, attend 3 hours of lecture each week, and take exams. Thus, the practicum students were able (and expected) to devote many more hours per week to the team project. This distinction is important to make because the SE students were not obligated to implement all the requirements specified in the original requirements document. Instead, they negotiated with their customers to agree upon the subset of requirements that they would implement.

4.1 Student Survey Validation of ADaPT

The survey, which was administered to Practicum and SE students, (see Appendix C.1) covered a wide range of issues, including template quality; perceived customer satisfaction; process activities and project success. There were thirty-eight respondents in the SE course; however, one of these was discarded because the respondent failed to complete the survey. There were nine respondents each from CSC 591O and BUS 516 (for a total of 18 students from the practicum).

In the practicum course, nine students had over five years experience in software engineering and development; some students also had extensive project management experience. In the SE course, students' experience in software development varied from one to five years. To account for these differences, the survey results are separated into four categories: e-commerce practicum students (*n*=9) with computer science background and extensive software engineering experience, e-commerce practicum students (*n*=9) with management background and limited software engineering experience; practicum students (*n*=18) as a group, and SE students (*n*=37) with computer science background and varying levels of programming experience. In the table below we specify the survey statement, and indicate the number of respondents and percentage (in parenthesis) of total students who agreed or strongly agreed to each statement for all four categories. We examined various factors including: template usefulness, ADaPT's effectiveness in comparison with other models, ADaPT's planning phase activities, ADaPT's implementation phase activities, and the project outcome. We measure the "usefulness" of a document or a process activity based upon the students' response to specific questions regarding a document or activity (see Survey in Appendix C.1).

The examination of the survey responses enabled us to conduct a preliminary validation of ADaPT's techniques for ensuring the development of a timely, high quality system with minimal planning and documentation while ensuring that RE best practices were applied. The analyses of each group of survey questions are summarized in the following subsections. We now present our actual survey results and discuss our analysis for each of the categories.

4.1.1 Usefulness of Templates

Students agreed or strongly agreed on the usefulness of the following templates:	CSC 591O Students (*n*=9)	BUS 516 Students (*n* =9)	All Practicum Students (*n*= 18)	CSC 326 Students (*n*=37)
Project Plan template.	6 (66%)	4 (44%)	10 (55%)	21 (57%)
Requirements template.	5 (55%)	3 (33%)	8 (44%)	20 (55%)
Requirements guide.	7 (78%)	4 (44%)	11 (61%)	23 (62%)

Analysis: The project plan and requirements templates were better received by computer science students than business students. Among all registered computer science students including practicum and

CSC 326, an average of 61% acknowledged that project plan template was useful, 55% acknowledged that requirements template was useful, and 70% agreed that requirements guide was helpful; fewer students with management background concurred on the utility of the templates. We attribute this difference to a possible lack understanding of the entire software development process and the importance of all product artifacts. This may be improved via additional training, which students expressed a desire for at the end of the semester. The data suggests that management students may require more process training than computer science students; we plan to prepare a more detailed training guide to enable students to employ the templates more effectively in the future.

4.1.2 ADaPT in Comparison to Other Process Models

Students agreed or strongly agreed to the following survey statements:	CSC 591O Students (*n*=9)	BUS 516 Students (*n* =9)	All Practicum Students (*n*= 18)	CSC 326 Students (*n*=37)
Software process model is essential for system development.	8 (88%)	8 (88%)	16 (88%)	21 (57%)
Familiarity with evolutionary models	8 (88%)	5 (55%)	13 (72%)	23 (62%)
ADaPT reduces planning and documentation in comparison to other models.	7 (78%)	4 (44%)	11 (61%)	18 (48%)
System requirements changed during project lifecycle.	8 (88%)	7 (78%)	15 (83%)	34 (91%)
ADaPT handles evolving requirements effectively.	7 (78%)	3 (33%)	10 (55%)	21 (57%)

Analysis: Computer science graduate students possessed more familiarity (88%) with evolutionary models than both computer science undergraduates and graduate management students. This partially explains why computer science graduate students noted that ADaPT reduces planning and documentation in comparison with other models. One can infer that undergraduate computer science and graduate management students were not able to adequately compare ADaPT to other software process models and methodologies due to their more limited experience. The graduate computer science students agreed by 88% that their project requirements changed during the project lifecycle; of these students 78% agreed that ADaPT handles changing requirements effectively. This is a positive indication that among students with strong software engineering backgrounds, ADaPT is believed to adequately support requirements evolution. Not surprisingly, 87% of all students (experienced and inexperienced) noted that their system requirements changed extensively during the project lifecycle. In the SE course, the requirements were intentionally changed throughout the semester to provide students with a more realistic project experience since requirements are known to be highly volatile in rapid development environments [CAD01].

4.1.3 Usefulness of Planning and Design Phase Activities

Students agreed or strongly agreed that the following ADaPT planning and design activities were useful:	CSC 591O Students (*n*=9)	BUS 516 Students (*n* =9)	All Practicum Students (*n*= 18)	CSC 326 Students (*n*=37)
Gathering requirements as scenarios.	9 (100%)	7 (78%)	16 (89%)	31 (83%)
Goal-Scenario Analysis.	8 (88%)	6 (66%)	14 (77%)	31 (83%)
Forming Subsystems helped in implementation planning.	8 (88%)	6 (66%)	14(77%)	27 (72%)
The planning phase enabled with a clear set of requirements.	8 (88%)	8 (88%)	16 (88%)	27 (72%)
Whiteboard and High-level architecture model was sufficient for design discussions.	9 (100%)	6 (66%)	15 (83%)	19 (51%)
Planning phase was accomplished in 3 weeks.	5 (55%)	5 (55%)	10 (55%)	15 (40%)
Risk analysis meetings helped identify and resolve conflicting requirements and design issues.	8 (88%)	6 (66%)	14 (77%)	25 (68%)

Analysis: An overwhelming majority of students agreed upon the importance of the planning phase. Among all students, 80% (on average) agreed that the planning phase enabled them to start the project with a clear set of requirements. The number of students that agreed upon the usefulness of scenario analysis in elaborating requirements was quite strong with 89% of the practicum students and 83% of the SE student responding favorably. 100% of the computer science graduate students agreed upon the usefulness of gathering requirements as scenarios. We believe that experienced software engineers understand and appreciate the usefulness of requirements engineering practices and can envision the benefits of these practices in software development [Bro95]. It is also likely that these students experienced a kind of "second-system" effect in that they had first hand knowledge of how difficult it is to "get the requirements right" using more traditional requirements analysis techniques. Roughly 67% of all students (in both classes combined) agreed that whiteboard and high-level architecture model was sufficient for documenting design. The students in SE class did not use whiteboards to support design discussions because they did not have access to an electronic whiteboard as did the practicum students. Instead, the SE students documented their designs using sequence and/or collaboration diagrams [Fow01b] as well as object oriented architecture models. A majority of practicum students (83%) agreed that their whiteboard design-discussions were effective. Although the planning phase is very imperative, an average of 50% of all students acknowledged that the planning phase was not accomplished in three weeks. Only 37% of all students felt that initial planning was minimal. Some projects were significantly more ambiguous and difficult to design/implement than others. Thus, due to the complexity of some systems, it was clear that more time should have been allocated to planning from the onset. This conflicts with our initial hypothesis that planning could be minimized and planning time shortened by using the ADaPT. In the future, we plan to allow more time for planning, especially for complex systems. The risk analysis meetings enabled the students to identify conflicting requirements and address design constraints. During the risk analysis meetings, roughly 73% of all students identified and resolved conflicting requirements, and addressed design constraints. The SE course instructor noted that the student teams who held regular risk analysis meetings were more successful in identifying conflicts among requirements and the privacy/security requirements for early resolution than the teams that did not hold regular risk analysis meetings.

4.1.4 Usefulness of Implementation and Testing Phase Activities

Students agreed or strongly agreed that the following ADaPT implementation and testing activities were used	CSC 591O Students (n=9)	BUS 516 Students (n =9)	All Practicum Students (n= 18)	CSC 326 Students (n=37)

and/or useful:				
Pair programming for implementation.	3 (33%)	3 (33%)	6 (33%)	19 (51%)
If yes, pair programming was effective.	3 out of 3 (100%)	3 out of 3 (33%)	6 out of 6 (100%)	17 out of 19 (89%)
Unit testing performed frequently.	8 (88%)	4 (44%)	12 (66%)	20 (54%)
System testing team performed system testing frequently.	9 (100%)	7 (78%)	16 (88%)	25 (67%)
Integrating subsystems was not difficult.	9 (100%)	4 (44%)	13 (72%)	15 (40%)
Writing acceptance tests before implementation improves quality.	7 (78%)	5 (55%)	12 (66%)	13 (35%)
Customer validation was performed at the end of the cycle.	6 (66%)	2 (22%)	8 (44%)	4 (10%)
Customer provided feedback frequently.	5 (55%)	3 (33%)	8 (44%)	9 (24%)
Customer is satisfied.	8 (88%)	7 (78%)	15 (83%)	10 (27%)

Analysis: Among the practicum students, 33% of the students practiced pair programming for implementation and almost all the students who used pair programming agreed upon its usefulness. Among the undergraduate SE students, 50% of them opted to use pair programming for implementation and 89% of them acknowledged the practice to be effective. We attribute the greater number of SE students that engaged in pair programming to the fact that the students in the SE course received pair programming training during a lecture that was devoted to this topic. In contrast, students in the practicum received no pair programming training. An average of 77% of all students agreed upon the usefulness of system testing by the system testing team and 50% of all students agreed that writing acceptance tests before implementation could improve system quality. Testing practices varied among the SE and practicum students. SE students were time constrained since they had homework, project, and exams as part of the course objectives. Additionally, since the professor was away for several weeks at the beginning of the semester, the students were assigned their projects rather late in the semester; these students did not have sufficient time at the end of the semester to apply all testing practices. In fact, at the end of the semester, the students expressed a desire to cover testing earlier in the semester so that they could devote more time to this activity in future projects. Because the practicum students, had only one course objective, to develop a system for the customer; the practicum students wrote acceptance tests before implementation. We attribute the varied testing practices in both courses to the survey results as discussed above.

The SE undergraduate students found system integration to be somewhat challenging; however, 100% of well-trained SE students acknowledged that integration was not a difficult task. It is believed that encouraging students to pair program in the future will improve this process; pair programming enables students with less experience to learn from experienced students and also reduces communication overhead costs [WKC00]. The survey has highlighted the need for a “System and Subsystem Testing” training module; this will form part of our plans for future work.

A meager average of 35% of all students agreed that customers provided frequent feedback. Among all practicum students, 83% agreed that their customer was satisfied with the validated system. Only 27% of the students in the software engineering course agreed that their customer was satisfied with the system. The remaining 75% were indifferent of customer satisfaction because they did not receive timely customer feedback. Although, ADaPT requires customer cooperation throughout the project lifecycle, the survey results show that the model does not heavily rely on timely customer input for successful delivery of the system. Analysts also realize that in reality frequent customer feedback is not always plausible, but hope to address this issue in future revisions to ADaPT.

4.1.5 Project Outcome using ADaPT

Students agreed or strongly agreed to the following survey statements:	*CSC 591O Students (n=9)*	*BUS 516 Students (n =9)*	*All Practicum Students (n= 18)*	*CSC 326 Students (n=37)*
Confident about product release.	9 (100%)	9 (100%)	18 (100%)	23 (62%)
Project is a success.	9 (100%)	9 (100%)	18 (100%)	28 (75%)
Initial planning was minimal.	6 (66%)	0	6 (33%)	12 (32%)
Product delivered on time.	9 (100%)	8 (88%)	17 (94%)	26 (70%)

Analysis: 100% of the practicum students agreed that their project was a success and 94% delivered their system on time. Roughly, 75% of the undergraduate SE students agreed their project was a success. 68% of undergraduate SE students believe that initial planning and documentation was excessive, as did 100% of the practicum management majors. This suggests that those students with more extensive software engineering experience have a greater appreciation for the benefits of RE, design, and planning activities than those students with limited experience. Future revisions to ADaPT will include training to ensure students understand the value of software engineering best practices and principles.

4.2 Instructor Survey Validation of ADaPT

A survey was administered to obtain instructors' perceptions for both the practicum and software engineering courses (see Appendix C.2). The survey covered instructors' opinions on a wide range of issues including: team cooperation, team communication, customer satisfaction, quality of the designed system and on time product delivery. The table below represents the responses of both instructors. The practicum instructor supervised four project teams and the SE instructor supervised nine project teams.

Students agreed or strongly agreed to the following survey statements about the project teams:	Practicum Teams: (*n*=4)	SE Project Teams (*n*=9)
Successfully completed the project.	4 (100%)	9 (100%)
Met its course requirements.	4 (100%)	9 (100%)
Functioned effectively resulting in a highly satisfied customer.	4 (100%)	9 (100%)
Developed a high-quality system.	3 (75%)	9 (100%)
Developed a system, which met all customer requirements.	3 (75%)	9 (100%)
Communicated effectively amongst team members.	3 (75%)	5 (55%)
Group was organized.	2 (50%)	6 (66%)
Cooperation amongst team members was good.	3 (75%)	5 (55%)
Received timely and frequent customer feedback.	1 (25%)	6 (66%)

Analysis-Practicum: The practicum instructor strongly agreed that all project teams completed the project successfully, met its course requirements, and ensured a highly satisfied customer. The instructor evaluated the progress of four teams; three teams successfully completed their projects on time and one team extended its work by two additional weeks beyond the end of the semester. Of the three projects successfully completed, the instructor strongly agreed that all teams developed a high-quality system, which met all customer requirements. According to the instructor, only one team (out of four) received any customer feedback; three teams developed their systems with little or no customer feedback. This suggests that ADaPT works well in situations where customer feedback is infrequent.

Analysis-CSC 326: The CSC 326 instructor agreed that all nine teams successfully completed their projects, met all course requirements, satisfied their customer, developed a high quality system, and developed systems that met all customer requirements. Only five teams communicated and cooperated effectively amongst their team members. The instructor also agreed that six teams demonstrated good organizational skills and received timely feedback. The teams that provided and received customer feedback also developed high quality systems; the instructor strongly agreed that three teams that developed high quality systems explicitly sought weekly customer feedback and allowed customers to evaluate prototypes throughout the semester.

4.3 Customer Survey Validation of ADaPT

A survey was used to obtain customer perceptions; these customers were official sponsors of the four Practicum projects and the nine SE projects (Appendix C.3). The survey addressed customers' opinions on various issues including: team-cooperation, team-communication, system requirements, systems' quality, on time product delivery, ADaPT artifacts and ADaPT methodology. Three customers participated in the survey and one customer has agreed to provide in-depth feedback in the next few weeks.

Customers agreed or strongly agreed to the following survey statements about the project teams:	Practicum Teams: (*n*=3)	CSC 326 Teams: (*n*=9)
Successfully completed the project.	3 (100%)	9 (100%)
Delivered the system on time.	3 (100%)	9 (100%)
Incorporated all requirements in the delivered system.	1 (33%)	9 (100%)
Developed a high-quality system.	3 (100%)	9 (100%)
Produced useful and essential artifacts.	3 (100%)	9 (100%)
Provided frequent progress reports on the project.	3 (100%)	8 (88%)

Analysis-Practicum: The Practicum sponsors agreed that their project teams successfully completed their project, developed a high quality system, delivered their system on time, and produced useful and essential artifacts (see Appendix D.1 for project descriptions). Three sponsors (*n*=3) (Art

Museum of NC, Lipsinc, NCSU) participated in the survey; one of the sponsors (Art Museum) strongly agreed that their project team had incorporated all customer-requirements in the final system. Lipsinc acknowledged that not all requirements were incorporated in the system delivered to them; however, this is because the team could not incorporate all system requirements due to Non Disclosure Agreement (NDA) issues between the university and Lipsinc. Thus, in the case of the Lipsinc system, the inability to incorporate all requirements was hindered by legal contracts rather than the process model employed for the project. The conference registration team, whose project was sponsored by NCSU, did not incorporate all system requirements on their customer's request; the customer for this group was satisfied with the system developed even though the requirements were not incorporated. Interestingly, the customer noted that this group had solved the most challenging and complex design problem for the system, the implementation of SSL for secure credit card transactions; this accounted for the customer being highly satisfied with the system. All three sponsors agreed that their respective project teams provided frequent progress reports. However, the sponsors for the practicum projects expressed indifference to the statements, which addressed the communication, cooperation and organizational skills of the team members. Based on the customer feedback provided via surveys, we believe that ADaPT is suitable for e-commerce system development; ADaPT enables teams to develop a high quality system to be delivered on time.

Analysis-CSC 326: The customer for all nine projects in the SE class agreed that the teams completed the project successfully, developed a high quality system, delivered their system on time, met all system requirements and produced useful and essential artifacts (see Appendix D.2 for project descriptions). It is important to note that the customer qualified their interpretation of the statement "met all system requirements" as follows: the customer indicated that the project teams negotiated with the customer to reach an agreement regarding those requirements they would be expected to incorporate by the end of the semester. The CSC 326 project teams had specified many more requirements than they would actually be able to implement; once all "desired" requirements were expressed, the students negotiated with the customer by agreeing to implement those requirements that were of highest priority to the customer. Thus, the customers only considered those requirements that had been mutually agreed upon when responding to this statement in the survey. The customer also observed that five teams demonstrated good organizational skills and appeared to cooperate well amongst their team members. Six of the nine teams communicated well and these teams provided frequent progress reports to their customer. The customer also acknowledged that some groups met weekly with her to provide project status reports, allowing frequent feedback on the prototype implementation over the course of the semester. These groups produced higher quality systems than those groups that did not meet with the customer on a regular basis. We thus recognize the development of high quality systems to be greatly influenced by the timeliness and frequency of customer feedback.

4.4 Summary

This chapter describes the validation efforts for ADaPT. The validation effort included the survey results to gather students', instructors' and customers' opinion on ADaPT. The survey results were comprehensively analyzed to measure the benefits and drawbacks of ADaPT. Finally, the lessons learned from validation were addressed as well. Chapter 5 discusses our plans for future work.

Chapter 5 Discussion and Future Work

The difference between the right word and the almost-right word is

the difference between the lightning and the lightning-bug. - Mark Twain

---

During the course of this thesis research, a number of observations were made which led to the development of additional opportunities for additional related work. Lessons learned from validation efforts as mentioned in chapter 4 include:

- ADaPT aims to minimize initial planning and documentation.
- An overwhelming majority of students delivered their system on time.
- The planning phase enables teams to start a project with a clear set of requirements.
- Using scenario analysis to identify missing requirements proved beneficial.
- Writing acceptance tests before implementation improves system quality.
- The system testing team supports ongoing testing throughout the project lifecycle as opposed to system testing performed only upon the implementation completion and prior to system delivery. Frequent system testing reduces bugs at the culminating stage and improves delivery speed.
- The provided templates were determined to be beneficial; an additional training guide is required to complement lack of training in software engineering amongst inexperienced students.
- Additional classroom lectures on software engineering process models are needed to improve understandability of inexperienced students.
- Additional refinement of templates will be useful.

- Our validation results demonstrate that customer feedback is not always timely and frequent; ADaPT has proven to work well under circumstances where customer feedback is often infrequent.

5.1 Conclusions

Fowler argues that the RE community often loses sight of the fact that requirements should be modifiable [Fow00]. He claims that software methodologies used should adapt to changing market requirements, while maintaining delivery speed. Several studies have shown that the majority of software errors can be traced to incorrect or misunderstood requirements [Boe81, End75, Lev86]. However, requirements evolution is particularly challenging in emerging application domains, such as e-commerce, in which the stakeholders often do not understand their own requirements. In reality, stakeholders often refine the understanding of their own requirements throughout a product's evolution. Given the prevalence for and consequences of misinterpreted and overlooked requirements, it seems that as a community we stand to either gain or lose a great deal from agile methodologies that claim to improve upon traditional requirements processes, but fail to actually do so.

Although agile methodologies offer improved delivery speed and adaptability, they fail to properly support RE practices. ADaPT aims to achieve a compromise between heavy and agile methodologies by documenting only essential requirements and planning artifacts. The goals of ADaPT are to: (1) introduce better requirements practices; (2) improve development speed, (3) minimize cost and (4) improve quality.

ADaPT uses goal and scenario analysis to elaborate requirements. In ADaPT, quality is strengthened with prototype acceptance testing and via risk analysis meetings, to ensure continual evaluation of a system's requirements throughout the project lifecycle by the stakeholders. Quality is further achieved via pair programming and rigorous testing. Our validation efforts, which began in January of this year, show that using ADaPT improves development speed and quality due to the focus on sound RE practices.

In summary, ADaPT is based upon four solid elements: a firm CMM basis for maturity; inclusion of proven RE best practices; adaptive prototyping to accommodate flexibility and speed; and a comprehensive risk analysis component combined with a thorough testing strategy to ensure software quality and process reliability. The ADaPT aims to develop high quality software, minimize software cost, improve development speed; and provide much needed support for requirements practices during rapid adaptive software development.

Case studies have served as the primary mechanism for validating the ADaPT. Validation was achieved via ADaPT's use in several e-commerce development projects for IBM, NCSU, Lipsinc, and a regional art museum within the NCSU Electronic Commerce Studio and in an undergraduate software engineering course. The validation results from the thirteen projects demonstrate that ADaPT is effective for rapid software development. It is important to note that the data collected from these student-run projects are as relevant as data collected in industry-run projects. Previous studies have shown that specifications produced by industry experts, under similar time constraints and pressure, are just as likely to be laden with ambiguities and conflicts [ACD01].

Component-based prototyping is effective for developing large and complex systems [Hig98]. In ADaPT, systems are developed in small manageable fractions (subsystems); therefore it will presumably work well for developing large complex systems. The ADaPT requirements workbook template ensures that developers list operational, privacy, and security goals categorically so that requirements pertaining to privacy and security policies may be tracked and assessed for compliance early on. Our future plans involve additional validation of the model, including a study of the model in relation to other established agile methodologies and the appropriateness of the model's artifacts in terms of level of formality, consistency, completeness, and density. We now discuss our related future work in the section below.

5.2 Future Work

The areas identified as opportunities for future work include: refinement of ADaPT, modification of templates, provision of additional training guide for inexperienced students, and in-depth analysis and comparison of ADaPT with evolutionary models.

Although our validation efforts demonstrate ADaPT's effectiveness and acceptance amongst students, certain modifications are in order. A majority of students admitted they were unable to complete the planning phase within three weeks. Although, this is no real surprise, we plan to address this issue by providing more upfront time for planning. Although pair programming has proven effective amongst the students who sought to use it for implementation, our results indicate that approximately 35% of all students actually used pair programming for implementation. Our future work will encourage students to use pair programming and perform in-depth analysis to determine the effectiveness of pair programming. The model will also be modified to enable students to interact with customers more effectively.

The modified templates and additional template-guides should enable inexperienced students to use the templates more easily. The students in the College of Management believe that the project plan template does not address every aspect of product management. The refined template will incorporate additional features, to address these additional product management issues. The students in College of Management failed to comprehend of the importance of the requirements workbook template and would benefit from additional guidance regarding its usefulness.

Researchers at NCSU have been performing extensive research in the areas of evolutionary prototyping and agile methodologies. As part of our future work, we plan to perform additional validation of ADaPT in different settings and conduct a comparison between ADaPT, evolutionary prototyping, and other agile methodologies. We plan to conduct a comprehensive analysis of these methodologies and their validation results in the near future.

5.3 Summary

In summary, ADaPT is based upon four solid elements: a firm CMM basis for maturity; inclusion of proven RE best practices; adaptive prototyping to accommodate flexibility and speed; and a comprehensive risk analysis component combined with a thorough testing strategy to ensure software quality and process reliability. ADaPT aims to develop high quality software, minimize software cost, improve development speed; and provide much needed support for requirements practices during rapid adaptive software development.

Bibliography

[Ale00] I. Alexander. The limits of eXtreme Programming, *IEEE Software*, 2000.

[AAB99] T.A. Alspaugh, A.I. Antón, T. Barnes and B. Mott. An Integrated Scenario Management Strategy, *IEEE 4th Int'l Symp. on Requirements Engineering (RE'99)*, Ireland, pp. 142-149, 7-11 June 1999.

[ADS00] A. Anton, J. Dempster & D. Seige. Deriving Goals from a Use-Case Based Requirements Specification for an Electronic Commerce System, *Proceedings of REFSQ*, 2000.

[AE00] A.I. Antón & J.B. Earp. A Multidisciplinary Electronic Commerce Project Studio for Secure Systems, *4th National Colloquim for Information Systems Security Education (NCISSE)*, Washington, D.C., May 2000.

[AE01a] A.I. Antón & J.B. Earp. Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems, in *Recent Advances in Secure and Private E-Commerce*, Kluwer Academic Publishers, pp 29-36, 2001.

[AEP01b] A.I. Antón, J.B. Earp, C. Potts & T.A. Alspaugh. The Role of Stakeholder Privacy Values in Requirements Engineering, *IEEE Int'l Symposium on Requirements Engineering (RE'01)*, pp 138-145, 2001.

[ACE01] A.I. Antón, R.A. Carter, J.B. Earp & L.A. Williams. *EPRAM: Evolutionary Prototyping Risk Analysis & Mitigation (e-Commerce Software Development Process Document)*, NCSU Technical Report, TR-2001-08, August 20, 2001.

[ACS01] A.I. Antón, R.A. Carter, H. Srikanth, A. Sureka, L.A. Williams, K. Yang, L. Yang. *Tailored CMM for a Small e-Commerce Company- Level 2: Repeatable*, NCSU Technical Report, TR-2001-09, August 23, 2001.

[Ant96] A.I. Antón. Goal-Based Requirements Analysis, S*econd IEEE International Conference on Requirements Engineering (ICRE '96)*, Colorado Springs, Colorado, pp. 136-144, 15-18 April 1996.

[Ant97] A.I. Antón. Goal Identification and Refinement in the Specification of Software-Based Information Systems, PhD Thesis, Georgia Institute of Technology, 1997.

[AP98] A.I. Antón and C. Potts. The Use of Goals to Surface Requirements for Evolving Systems, *Int'l Conf. on Software Engineering (ICSE '98)*, Kyoto, Japan, pp. 157-166, 19-25 April 1998.

[Ayo96] M. Aoyama. Componentware: Building Applications with Software Components, *J. of IPSJ*, Vol. 37, No. 1, pp. 71-79, 1996.

[Ayo97] M. Aoyama. Process and Economic Model of Component-Based Software Development, *IEEE SAST (Symposium on Assessment of Software Tools)*, June 1997.

[BEK96] T.Bardo, D.Elliot, T. Krysak, M. Morgan, R. Shuey, W. Tracz. A Product Line Success Story, Crosstalk: *The Journal of Defense Software Engineering*, 1996.

[BW84] V. Basili & D. Weiss. A Methodology for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering*, Vol. 10., November 1984.

[Bea99] R. Beaumont. Information Systems Development Methods. Source: *Laptop*; C;/Hicourseweb new/chap12/slide3/dest1.doc, 1999.

[Bec00] K. Beck. *Extreme Programming Explained*, Addison-Wesley, 2000.

[Bec99] K. Beck. Embracing Change with eXtreme Programming, *IEEE Computer*, Vol.32, No. 10, October 1999.

[Bro95] F. Brooks. *The Mythical Man-Month,* Anniversary Edition, Addison Wesley,1995.

[Bec00] K. Beck. Emergent Control in eXtreme Programming, *E-Project Management Advisory Service*, 2000.

[BDS00] M. Beedle, M. Devos, Y. Sharon, K. Schwaber & J. Sutherland. SCRUM: An Extension Pattern Language for Hyper-productive Software Development, 2000.

[Boe88] Boehm, B. A Spiral Model for Software Development and Enhancement, *IEEE Computer*, 21, pp. 61-72, 1988.

[BI96] B. Boehm & H. In. Identifying Quality-Requirements Conflicts, *IEEE Software*, 13 (2), pp. 25-35, March 1996.

[Bol00] T. Bollinger. XP: Two Concerns, *IEEE Software*, 2000.

[Bra00] T. Bragg. eXtreme Programming Enterprise, *E-Project Management Advisory Service*, October 2000.

[Car01] R. Carter. EPRAM*: A Risk Analysis and Mitigation-Based Evolutionary Prototyping Model for Quality Requirements Development*, M.S. Thesis, North Carolina State University, May 2000

[CAD01] R. Carter, A. I. Anton, A. Dagnino & L. Williams. Evolving Beyond Requirements Creep: A Risk Based Evolutionary Prototyping Model, *IEEE Int'l Symposium on Requirements Engineering,* pp 94-101, Toronto, Canada, August 2001.

[Coc99] A. Cockburn. *Software Development as a Cooperative Game,* Humans and Technology, Inc., 1999.

[Coc00] A. Cockburn. *Balancing Lightness with Efficiency,* Cutter Consortium, September 2000.

[CW00] A. Clouse, C. Wells. *Transitioning from EIA/IS-731 to CMMI*, July 2000.

[CW00] A. Cockburn and L. Williams. The Costs and Benefits of Pair Programming. *eXtreme Programming and Flexible Processes in Software Engineering XP*, 2000.

[Cri01] L. Crispin. Is Quality Negotiable, *XP Universe*, July 2001.

[CH01] L. Crispin & T. House. Testing in the Fast Lane*:* Automating Acceptance Testing in an Extreme Programming Environment, *XP Universe*, July 2001.

[CSW97] Cunningham. D, Subrahmanian. E, and Westerberg. A. User-Centered Evolutionary Software Development Using Python and Java, *Proceedings of the 6th International Python Conference*, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA, 1997.

[Dav92] A.M. Davis. Operational Prototyping: A New Development Approach, *IEEE Software*, 9(5), pp. 70-78, September 1992.

[DBC88] Davis. A, Bersoff. E, and Comer. E. A strategy for comparing alternative software development life cycle models, *IEEE Transactions on Software Engineering*, Vol 14, No 10, 1453-1461, 1988.

[DvLF93] A. Dardenne, A. van Lamsweerde and S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 201(1-2):3-150, April 1993.

[Els01] A. Elssamadisy. XP on a Large Project – A Developer's View, *XP Universe*, July 2001.

[FL90] N. Fenton, B. Littlewood. Software Reliability and Metrics, *Elsevier Applied Science*, 1990.

[FS01] M. Foulkrod & M. Silverstein. A Collaborative Model for Developers and Testers using the Extreme Programming Methodology, *XP Universe*, July 2001.

[Fow01a] M. Fowler. *The New Methodology,* ThoughtWorks Inc., November 2001.

[Fow01b] M. Fowler. *Analysis Patterns: Reusable Object Models*, First Edition, Addison Wesley, 1996.

[Gra89] I. Graham. Structured Prototyping for Requirements Specification of Expert Systems, *IEEE Colloquium on Expert Systems Lifecycle,* pp. 5/1-5/3, 1989.

[Gra92] R. Grady. Practical Software Metrics for Project Management and Process Improvement, Prentice Hall, Englewood Cliffs, NJ, 1992.

[Gre01] J. Grenning. Using XP in a Big Process Company: A Report from the Field, *XP Universe,* July 2001.

[Het93] B. Hetzel. Making Software Measurement Work: Building an Effective Measurement Program, John Wiley & Sons, 1993.

[Hig98] J. Highsmith. *Application Development Strategies: Managing Complexity,* Cutter Information Corp., 1998.

[Hig00] J. Highsmith. E-Business Application Delivery: eXtreme Programming, *The Monthly Journal on Developing and Delivering Applications in Today's E-Business Economy*, February 2000.

[Hig00] J. Highsmith. Using Adaptive Software Development to meet the challenges of a high-speed, high-change environment, *Software Testing and Quality Engineering Magazine*, July-August 2000.

[Hig01] J. Highsmith. Debating After Action Reports and Heavy versus Light Methods, *Cutter Consortium,* 2001.

[Hig01] J. Highsmith. Documentation is Not Understanding, *Cutter Consortium,* 15 March 2001.

[Hig01] J. Highsmith. Light Architecture, *Cutter Consortium*, 15 February 2001.

[Hig01] J. Highsmith. Methodologies and Requisite Variety, *Cutter Consortium*, 1 February 2001.

[HR00] S. Hawrysh, J. Ruprecht. Light Methodologies: It's Like Déjà vu All Over Again, *Cutter Consortium*, 2000.

[HS01] M. Hohman & A. Slocum. *Mob Programming and the Transition to XP*, July 2001.

[Hum00] W. Humphrey. Comments on eXtreme Programming, *IEEE Software*, 2000.

[Hum89] W. Humphrey. Managing the Software Process, *SEI Series in Software Engineering*, ISBN: 0-201-18095-2, 1989.

[Jac92] I. Jacobson. Object-Oriented Software Engineering: A Use Case Driven Approach, ACM Press, 1992.

[Jal00] P. Jalote. CMM in Practice – Processes for Executing Software Projects at Infosys, Addison-Wesley, 2000.

[JA00] J. Jarzombek, B. Allgood. Up Close with Lt. Col. Joe Jarzombek and Bruce Allgood, July 00.

[Jef00] R. Jeffries. eXtreme Testing: A Path to Rapid, Reliable Development, *E-Project Management Advisory Service*, 2000.

[JST01] K. Johansen, R. Stauffer, D. Turner. Learning by Doing: Why XP Doesn't Sell, *XP Universe*, July 2001.

[KC00] N. Kini, S. Collins. *Lessons Learned from an XP Project*, Tensegrent, USA.

[KC01] N. Kini, S. Collins. Steering the Car: Lessons Learned from an Outsourced XP Project, *XP*

Universe, July 2001.

[[Kir01] D. Kirkpatrick. Finding the Right Process Mix in the Real World, *XP Universe*, July 2001.

KHH00] J. Kivi, D. Haydon, J. Hayes, R. Schneider, *G. Succi. eXtreme Programming: A University Team Design Experience,* University of Calgary, Canada, 2000.

[Lam99] W. Lam. Managing Requirements in a Product Family Approach to Systems Engineering, John Wiley & Sons Inc., Syst Eng 2, pp 46-55, 1999.

[Lam00] A. Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective, *IEEE International Conference on Software Engineering (ICSE00),* 2000.

[Lam01] A van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour, *IEEE 5th Int'l Symp. on Requirements Engineering (RE'01)*, Toronto, Canada, pp. 249-261, 27-31 August 2001.

[Lei01] J. Leite. Extreme Requirements, PUC Rio, Rio De Janeiro, Brazil, 2001.

[Lit01] J. Little. Up-Front Design versus Evolutionary Design in Denali's Persistence Layer, *XP Universe,* July 2001.

[Lov01] G. Lovaasen. Brokering with eXtreme Programming, *XP Universe,* July 2001.

[Mar00] R. Martin. eXtreme Programming Development through Dialog, *IEEE Software*, July/August 2000.

[Mel00] S. Mellor. Metaphors, Magic, War, Numbers, and The Elite, *IEEE Software,* 2000.

[MC01] R. Miller & C. Collins. Acceptance Testing, *XP Universe*, July 2001.

[MAN01] S. Mitchell, B. Auernheimer & D. Noble. Scenarios, Tall Tales, and Stories: Extreme Programming the Oak Grove Way, *XP Universe*, July 2001.

[OC99] S. Otoya & N. Cerpa. An Experience: A Small Software Company Attempting to Improve its Process, *Proc. Software Technology and Engineering Practice*, *STEP '99*, pp. 153-160, 1999.

[PJD01] A. Parrish, J. Jones & B. Dixon. Extreme Unit Testing: Ordering Test Cases to Maximize Early Testing, *XP Universe,* July 2001.

[Pau00] M. Paulk. XP from a CMM Perspective, *IEEE Software*, 2000.

[Pau98] M.C. Paulk. Using the Software CMM in Small Organizations, Joint 1998 Proc. Pacific Northwest Software Quality Conf. and the Eighth Int'l. Conf. on Software Quality, pp. 350-361, October 1998.

[Pau99] M.C. Paulk. Using the Software CMM With Good Judgment, *ASQ Software Quality Professional*, 1(3), pp. 19-29, June 1999.

[PCC93] M.C. Paulk, B. Curtis & M.B. Chrisis. *Capability Maturity Model for Software. Version 1.1*, Software Engineering Institute Technical Report, CMU/SEI-93-TR, February 24, 1993.

[Phi00] D. Phillips. Process/Anti-Process, *Cutter Consortium,* 16 August 2000.

[PS00] M. Philips, S. Shrum. *Creating an Integrated CMM for Systems and Software Engineering*, Software Engineering Institute, September 2000.

[Pie00] B. Pierce. *Is CMMI Ready for Prime Time*, Northern Utah Process Improvement Technologies, 2000.

[Pin01] S. Pine. An Application of XP in a multiple team/multi-process environment, *XP Universe*, July 2001.

[PH01] C. Poole, J. Huisman. Extreme Maintenance, *XP Universe*, July 2001.

[PTA94] C. Potts, K. Takahashi & A.I. Antón. Inquiry-Based Requirements Analysis, *IEEE Software*, 11(2), pp. 21-32, March 1994.

[Pou97] J. Poulin. *Measuring Software Reuse*, Addison Wesley, 1997.

[Pou95] J. Poulin. Software Reuse on the Army SBIS Program, Crosstalk: *The Journal of Defense Software Engineering*, 1995.

[Pre97] R. Pressman. *Software Engineering: A Practitioners Approach*, Fourth Edition, McGraw Hill, 1997.

[RS94] V. Rajlich, J. Silva. A case study of Software Reuse in Vertical Domain, *Proceedings of the 4th Systems Reengineering Technology Workshop*, Monterey, CA, 1994.

[RS96] V. Rajlich, J. Silva. Evolution and Reuse of Orthogonal Architecture, *IEEE Transactions on Software Engineering*, 1996.

[RR94] M. Ramesh, H. Rao. Software Reuse: Issues and an Example, *Decision Support Systems*, 1994.

[Ril01] J. Riley. *XP in Space*, March 2001.

[RJ00] L. Rising, N. Janoff. The Scrum Software Development Process for Small Teams, IEEE Software, 2000.

[Ros77] D. Ross & K. Schoman. Structured Analysis for Requirements Definition, IEEE Transactions on Software Engineering, Vol. 3, No.1, 1977.

[Roy90] W. Royce. Pragmatic Quality Metrics for Software Development Models, Proc. Conf. On TRI-ADA '90, pp. 551-565, 1990.

[Roy98] W. Royce. *Software Project Management: A Unified Framework*, Addison Wesley, 1998.

[Rup00] J. Ruprecht. Please, Not Another Methodology Feud, Cutter Consortium, 21 June 2000.

[Rus00] L. Russell. Heavy Versus Light Methods For Developing It: Business Solutions, E-Project Management Advisory Service, 2000.

[Sch00] K. Schwaber. Against a Sea of Troubles: Scrum Software Development, E-Project Management Advisory Service, 2000.

[SB01] K. Schwaber & M. Beedle. Agile Software Development with Scrum, Prentice Hall, 2001. [Sch01] G. Schalliol. Confessions and Complaints from a Team of XP Analysts, ThoughtWorks, Inc., July 2001.

[Sch01] G. Schalliol. Challenges for Analysts on a Large XP Project, *XP Universe*, July 2001.

[She00] C. Shelley. Our Collision with XP: What we Picked Up, *IEEE Software*, 2000.

[Shr99] S. Shrum. Spotlight: CMMI Model Representations, *SEI Interactive*, December 1999.

[Shr99] S. Shrum. Choosing a CMMI Model Representation, *Software Engineering Institute*, 1999.

[Sid00] J. Siddiqi. An Exposition of XP But No Position on XP, *IEEE Software*, 2000.

[Spi00] D. Spinellis. Taking Common Sense to the eXtreme, *IEEE Software*, July/August 2000.

[Som95] I. Sommerville. *Software Engineering*, Addison-Wesley, 1995.

[Sta00] R. Starbuck. A Configuration Manager's Perspective, July 2000.

[TF00] C. Taber, M. Fowler. An Iteration in the Life of an XP Project, *E-Project Management Advisory Service*, 2000.

[Wat00] J. Waters. eXtreme Method, Application Development Trends, July 2000.

[Wel01]. D. Wells. Extreme Programming: A gentle introduction, www.extremeprogramming.org, 2001.

[Wes01] D. West. Enculturating Extreme Programmers, XP Universe, July 2001.

[WBF00] J. Weszka, P. Babel & J. Ferguson. CMMI: Evolutionary Path to Enterprise Process Improvement, July 2000.

[WKC00] L. Williams, R. Kessler, W. Cunningham & R. Jeffries. Strengthening the Case for Pair Programming, IEEE Software, July/August 2000.

Appendix A ADaPT Process Description Guide

Introduction

ADaPT is software process model designed to improve the software quality and delivery speed of rapidly developed systems. ADaPT helps manage changing requirements yielding an adaptive software process. In today's fast-paced and competitive world of commercial software development, speed and flexibility are mandatory. Since the early 1970's, a large number of lifecycle models have been introduced. *Waterfall* model [Roy90] has been blamed for causing software to be more expensive, delivered later, more unreliable, and unable to address changing requirements [DBC88]. In the late 1980's *Evolutionary* lifecycle model was introduced, where the goal is to "evolve" or "grow" some or all of system's functionality into the final product iteratively [CSW97]. Although evolutionary approaches were iterative in nature, they incorporate mini-waterfalls within each development cycle [Hig98]. The evolutionary approaches involved significant documentation and planning. *Adaptive* prototyping lifecycles, which provides the basis for the ADaPT model, were introduced to reduce the length of product delivery [Hig98].

An overview of the ADaPT model and its phases are discussed herein. The ADaPT model's planning and design phase as well as implementation and testing phase are described in subsections 1.1 and 1.2, respectively. Section 2 provides a mini tutorial for applying ADaPT.

1 ADaPT Model

Two main phases comprise of ADaPT: (1) the planning and design phase and (2) the implementation and testing phase. During planning and design phase, developers elicit requirements in the form of scenarios and goals, and begin project planning. During implementation and testing phase, developers employ pair programming to write the code and conduct extensive testing. Figure 1 provides a high level overview of ADaPT. The ovals portray high-level project team activities and block lines represent quality assurance activities. ADaPT calls for basic essential documentation to ensure an adaptive process. Documentation in maintained in spreadsheets; each subsystem's documentation is written during the cycle in which it is developed. As previously mentioned, risks are evaluated during risk analysis meetings (as shown in Figure 1).

A more detailed overview of the ADaPT process is provided in Figure 2. The oval-shaped figures represent process activities, curved-rectangles represent the documentation artifacts, thicker arrows represent major control flows through the process, and the narrower arrows indicate data flowing because of the activities. The shaded ovals represent the processes that take place throughout the cycle. We now discuss each of these aspects of the model.

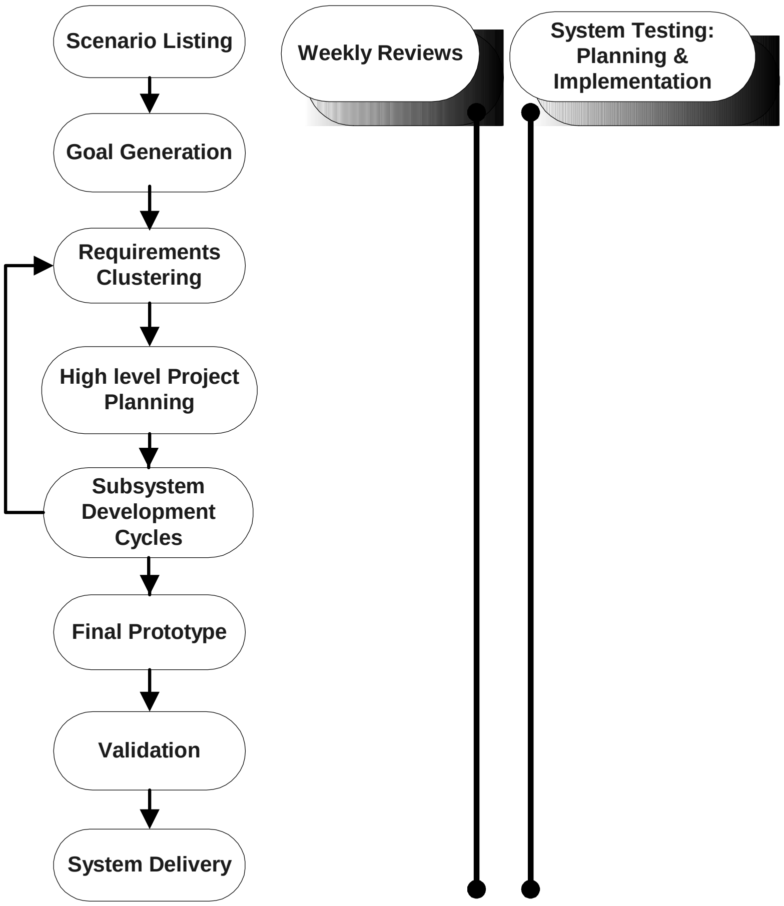


Figure 1: High Level view of ADaPT

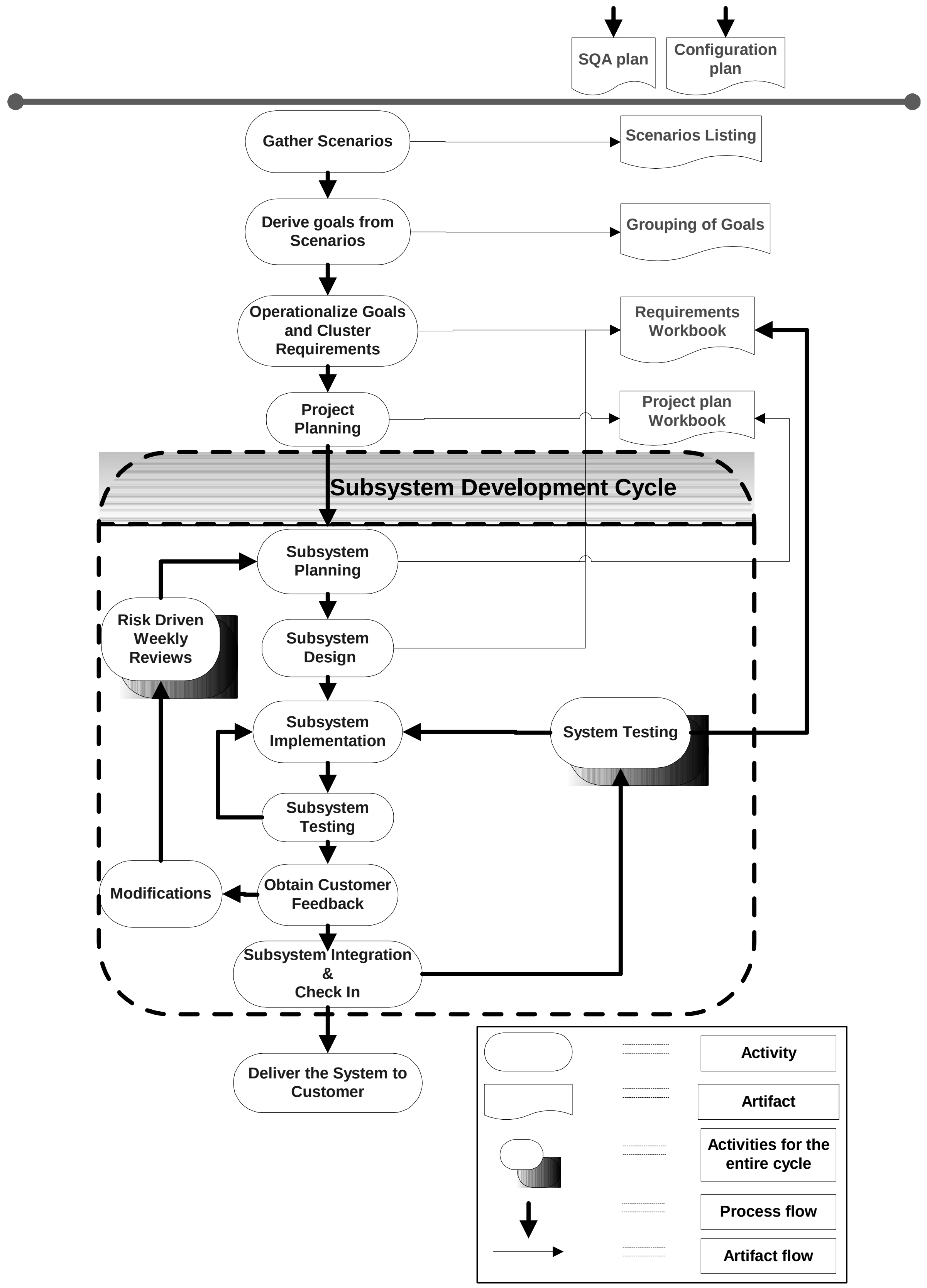


Figure 2: ADaPT Process Model

1.1 The ADaPT Planning and Design phase

During the planning and design phase, requirements are elicited in the form of scenarios and elaborated with goals before the development team begins actual project planning. Two artifacts are produced during this phase: a planning workbook and a requirements workbook. The requirements workbook documents the system requirements and the planning workbook documents the project plan and high-level subsystem design. The planning and design phase should take less than three weeks. Figure 3 graphically portrays the five planning phase activities. In this figure, "M" represents the mechanism involved in the process activity, "A represents the process activity itself and "D" represents the artifact or documentation produced by process activity. We now discuss these activities.

Activity 1: Scenario Analysis

An initial planning meeting between the stakeholders (e.g. customers and/or users) and the developers is held to gather information about the desired system. During this and subsequent meetings with stakeholders, scenarios that reflect the system as described by the customer/user are created. As few as five scenarios or as many as several hundred scenarios may be documented. Consider the following six scenarios for an online shopping site:

S_1: Search for products

S_2: Check product availability

S_3: Compare product features/price

S_4: Add item to the shopping cart

S_5: Register

S_6: Complete purchase

Scenarios such as these would serve as the basis for goal elaboration and subsystem design as we now discuss. For the remainder of this section we employ an online shopping site example to describe the ADaPT process activities.

Activity 2: Generate Goals

The collected scenarios are analyzed and later elaborated with goals that must be achieved. In ADaPT these goals ultimately represent operational requirements. Analysts may employ various techniques to analyze the scenarios such as the Inquiry Cycle Model [PTA94] or the GBRAM [AP98], but the objective is to generate goals to ensure scenario satisfaction. Using the GBRAM we elaborate two of the above scenarios with the goals required to satisfy each scenario as follows.

$S_{1:}$ Search for products

G_1: `(System) PROMPT user to enter search keywords`

G_2: `(System) SEARCH for keyword matches`

G_3: `(System) GENERATE search results web page`

G_4: `(System) DISPLAY search results web page`

$S_{3:}$ Compare product features/price

G_1: `(User) SELECT products to be compared`

G_2: `(System) SEARCH for product features`

G_3: `(System) GENERATE table with product comparison info`

G_4: `(System) DISPLAY search results web page`

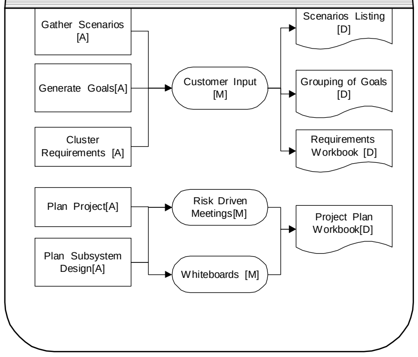
Note that each goal represents an event comprised of an actor/action tuple as in [AAB99]. Once the scenarios have been elaborated with goals, the goals are clustered according to Activity 3, below.

Activity 3: Cluster Requirements

The goals generated for each scenario are organized so that related goals form logical subsystems. Thus, subsystems are formed by clustering related goals; the approach is similar to the hierarchical approaches taken in [Ant97] and [DvLF93]. A subsystem is a fraction of the system and represents functionality that can be implemented independently.

The goals that form a subsystem may not necessarily come from one particular scenario. In other words, a subsystem can be comprised of some goals generated from S_1, some from S_3, and so on. In our example, goals G_1, G_2 and G_3 from S_1 may be grouped with goals G_1 and G_2 from S_3 to form a subsystem. All five goals address different kinds of searches and are thus clustered to form one subsystem, documented as a "search" subsystem in the requirements workbook. The documented subsystem and its respective requirements are revisited during the subsystem's implementation to ensure consistency and understandability. Only the requirements corresponding to the subsystem developed during a given cycle are listed and updated during that cycle. For XP practitioners, an ADaPT subsystem is roughly equivalent to an XP "user story" [Bec00].

Figure 3: ADaPT Planning and Design Phase



Activity 4: Plan Project

Once logical requirements are organized into subsystems, the subsystem with the highest priority (determined based on dependency) is developed first. The technical leaders responsible for individual subsystems are appointed. The project planning activity is characterized as high-level and encompasses estimating time and resources needed to develop the product; this information is documented in the planning workbook. The project plan provides: a brief system overview; a list of all team members and their respective roles; the prioritized list of subsystems; and the scheduled delivery date for each subsystem. During each cycle, subsystem planning is performed before its implementation, though one or more subsystems may be developed at a given time. Subsystem design is discussed below.

Activity 5: Design Subsystem

One or more subsystems are chosen for development during a given cycle (see Figure 2). The subsystem requirements are re-evaluated before development begins. Each subsystem is broken down into several tasks; the tasks are then documented in the planning workbook. These tasks are assigned to team members and the scheduled completion date for each task and the subsystem are documented accordingly. The design issues pertaining to a subsystem are discussed using electronic whiteboards and documented in the planning workbook. Copies of whiteboard drawings are maintained within the planning workbook. Design meetings focus solely on those subsystems being developed during a given cycle. Use-case diagrams may be used to show the design elements and inter-subsystem relationships.

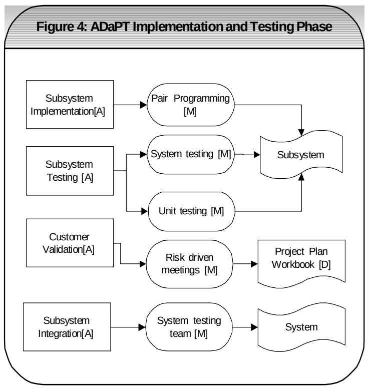
1.2 Implementation and Testing Phase

This section describes the ADaPT implementation and testing phase. Every subsystem is developed according to Figure 2. Figure 3.5 graphically depicts the four key activities that comprise the subsystem implementation and testing phase.

Activity 1: Subsystem Implementation

Subsystem development cycles are short and usually do not extend to more than a month. If a particular subsystem is estimated to take longer, it should be broken down into smaller subsystems. The goal is to focus on each small piece (subsystem) one at a time and do so thoroughly with proper planning and feedback from the customer.

Pair programming [CW00, WKC00] has worked well in situations where the requirements change frequently (e.g. e-commerce) and the projects are complex. In pair programming two developers work together, as they take turns in writing code. One developer observes the other developer writing code; but the resulting source code reflects both developers' ideas. ADaPT employs pair-programming during implementation as a technique to improve software quality.



Activity 2: Subsystem Testing

Another way in which quality is addressed in ADaPT is via subsystem and system testing. Subsystem testing entails white box testing, black box testing and acceptance testing. White box testing ensures that all program statements are executed, according to program structure. Black box testing focuses on program test cases that are based on the system specification. Acceptance tests ar tests conducted to enable the customer to validate that the requirements for each subsystem have been satisfied.

In ADaPT, subsystem tests may be written by the stakeholders before the subsystem implementation, and performed throughout the development cycle. System testing is performed throughout the lifecycle to ensure all system elements are properly integrated. At the end of each cycle, every subsystem is tested thoroughly and integrated with the other subsystems.

Activity 3: Customer Validation

The customer evaluates the system prototypes via acceptance tests to ensure their software requirements are met. Any customer-suggested modifications (such as new or missing requirements) must be addressed. The requirements workbook is updated with these requirements changes. Since the customer’s opinion is taken before, during and at the end of every subsystem cycle, requirements changes during final validation are expected to be minimal. The customer's validation at the end of every prototyping cycle enables developers to iteratively revisit the requirements and ensures risk minimization.

Activity 4: Subsystem Integration

The system testing team is responsible for integrating each subsystem with the overall system. The subsystems are integrated with other subsystems and deposited (checked-in) in a repository after the modifications suggested by the customer are completed and customer satisfaction is ensured.

2 Operational Example of ADaPT

In this section, we demonstrate the operation of the ADaPT model within the context of an ordinary Mark IV 20-cup coffee maker system. The average cost of this type of coffee maker is twenty dollars; thus, its functionality is rather simple. For convenience and ease of understanding, we elaborate only the system viewpoints while designing this coffeemaker.

The GQM paradigm (as mentioned in section 2) follows three steps [BW84]:

List all the major goals.

Derive *Questions* that are needed to determine if the goals are achieved or not. For every goal, define questions that have to be answered within the borders of the goal.

Decide what is to be measured in order to answer the questions.

In the example shown below, the goal is design a coffee maker and the scenarios are identified to ensure the satisfaction of each goal and the metrics are the requirements generated for the system. Figure 5 graphically depicts the results of applying the GQM process.

The terminal nodes are the requirements identified for the system (shown as R_x in Figure 5). Our objective is to design a simple Mark IV coffee maker and the process of identifying scenarios and generating requirements is discussed below. Applying traditional requirements engineering scenario analysis led to the process of identifying scenarios and goals in the process of manufacturing a coffee maker.

S_1: Brew some coffee

$G_{1.1}$ / R_1: *BOIL* water when control switch is "On".

$G_{1.2}$: *BOIL* water until water intake is empty.

R_2: *AVOID* boiling when there is no water.

R_3: *PREVENT* steam coming out of valve when boiling water.

$G_{1.3}$ / R_4: *BREW* water in water intake as coffee until water intake is empty.

$G_{1.4}$ / R_5: *SWITCH* indicator light "On".

S_2: Keep the coffee warm

$G_{2.1}$ / R_6: MAINTAIN warmer plate warm at 185^0F temperature.

$G_{2.2}$ / R_7: INTERRUPT warming when pot is removed.

$G_{2.3}$ / R_8: INTERRUPT warming when control switch is “Off”.

After the requirements are generated, they are grouped into subsystems to be developed in a subsystem cycle. The requirements for manufacturing the coffee maker can be grouped into three subsystems.

SS_1: BOIL Water functionality: R_1, R_2, R_3

SS_2: BREW button Functionality: R_4, R_5.

SS_3: WARMER PLATE Functionality: R_6, R_7, R_8.

These scenarios in the development of the Coffee Maker may also be described as “user stories” according to the practices of Extreme Programming [Bec00]. *User stories* are written to describe system needs by the customer [Bec00].

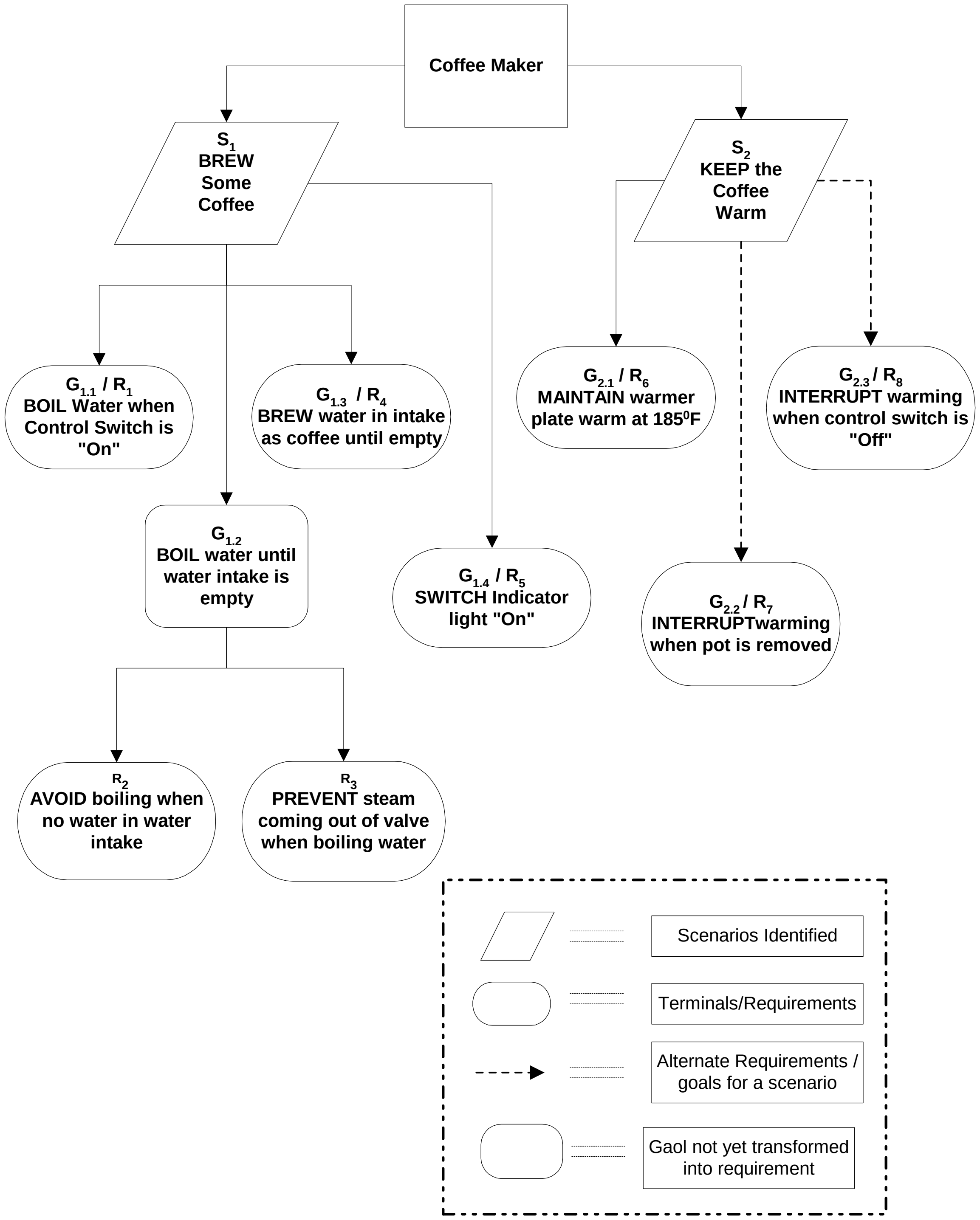

Coffee Maker
S1 BREW Some Coffee
S2 KEEP the Coffee Warm
G1.1 / R1 BOIL Water when Control Switch is "On"
G1.3 / R4 BREW water in intake as coffee until empty
G2.1 / R6 MAINTAIN warmer plate warm at 185⁰F
G2.3 / R8 INTERRUPT warming when control switch is "Off"
G1.2 BOIL water until water intake is empty
G1.4 / R5 SWITCH Indicator light "On"
G2.2 / R7 INTERRUPTwarming when pot is removed
R2 AVOID boiling when no water in water intake
R3 PREVENT steam coming out of valve when boiling water
Scenarios Identified
Terminals/Requirements
Alternate Requirements / goals for a scenario
Gaol not yet transformed into requirement

Figure 5: System Viewpoints for Manufacturing Coffeemaker

3 Risk Mitigation in ADaPT

One of the top ten management principles, according to Royce, is the establishment of an iterative life-cycle process that confronts risk early on [Roy98]. The ADaPT model addresses risks related to a project before the project starts. ADaPT supports *risk mitigation* via weekly risk-analysis meetings. The development team and testing team hold weekly reviews to discuss issues and maintain progress reports.

"Requirements creep" refers to significant modifications to the existing documented requirements for a software system throughout the lifecycle, resulting in extensions to and alteration of the software's functionality and scope [Car01]. Requirements creep, if any, is addressed in the weekly reviews. Risks related to the project are discussed during weekly reviews. The team members, along with other project issues, address the new risks, including addition of new requirements. Risks may also surface due to changes in application domain, problem domain and development environment [Car00]. During the weekly risk-driven meetings, risks are evaluated and the consequences pertaining to the risk are analyzed. Solutions to mitigate the risk are discussed, documented and worked upon.

4 Requirements Evolution in ADaPT

The system is developed in fractions and a small fraction (subsystem) of the system is developed every cycle. The customer takes part in prioritizing the subsystems and this priority drives the implementation of the system. The customer also provides acceptance tests, which the system must pass to meet his/her expectations. In ADaPT, the customer is actively involved before and during the development of the system.

The customer validates the system after the successful implementation of each subsystem. At the end of the subsystem development cycle, the customer evaluates the efforts put forth by the development team from the start of the project to date. The modifications, if any, are made to the system developed so far and deposited into the repository. Therefore, at the end of every cycle the customer's expectations are evaluated with the system at hand.

The ADaPT model ensures requirements changes are effectively managed. ADaPT minimizes requirements creep by involving the customer early on and throughout the life of the project.

5 Summary

In this chapter, we introduce the ADaPT process model for software development. ADaPT uses goal and scenario analysis to elaborate requirements by incorporating the concept of “user stories” in XP with scenarios as traditionally used in RE. In ADaPT, quality is strengthened with prototype acceptance testing and via risk analysis meetings, which ensure continual evaluation of a system’s requirements throughout the project lifecycle by the stakeholders. Quality is further achieved via pair programming and rigorous testing. We believe our current validation efforts, which began in January of this year, will show that using ADaPT improves development speed and quality due to the focus on sound RE practices.

Appendix B Templates

B.1 Project Planning Workbook Template

Project Plan Author:
Project Plan Owner:
Initial Version
Last Updated
Version #

Provide System Overview

Team Members and Project Roles					
Team Member	Title	Roles	Responsibilities	Role Description	

Description of Subsystems for the project					
Subsystem No.	Subsystem Description	Subsystem Priority	TL responsible	Work Weeks required	Target Completion Date

Document Revision History

Terms used:	
WW required:	The number of workweeks required to complete the implementation of this subsystem.
Target Completion Date	The Scheduled delivery/validation of the subsystem to the customer.
TL Responsible	Technical lead responsible to overlook the implementation of the subsystem. A team can have more than one technical lead depending on the size of the project.
Subsystem Priority	Prioritize subsystems for implementation based on the importance of the subsystem as part of the system

Sub tasks	Each requirement can be broken down into several sub-tasks and assigned to one or more team members; e.g. a requirement "Display Navigation Menu" can have sub-tasks: "Determine the possible links in the Navigation menu", "Determine the location to place the menu", Determine the background and the text color for the links and so on.

Provide Subsystem 1 Overview

Subsystem 1								
Subsystems	Description	TL Responsible	Requirements [provide ref #]	Sub-Tasks	Member Responsible	WW Done	WW Remaining	Target Completion Date

Document Revision History

Description of terms used

WW Done	WW done [Work Weeks Done]: The number of man weeks of work completed from the start of the subsystem. E.g. At the beginning of the subsystem, the work weeks done is zero. If four weeks is assigned for the development of subsystem 1; at the end of completion of one week, WW done = 1 and WW remaining = 3.
Subtasks	Break down of the subsystems requirements into sub tasks and assignment of the subtasks to the developers performed by TL

B.2 Requirements Workbook Template

System Overview	
Project Name-Document Name	
Version Number	
File Name	
Revision Date	
Document Author	
System Overview	Provide a brief overview of the system that is covered by the specification.
Goals to be achieved from this project	Provide a list of expectations of the new system, both in terms of what must be improved and what must be retained from the current processes.
Glossary	

Project Requirements									
Requirement ID	Requirement Description	Subsystem number	Type (e.g. Functional, privacy, security etc)	Criticality	Technical issues	Cost and Schedule	Risks	Dependencies with other requirements	Others

Subsystem-1 Requirements							
Subsystem Overview: Provide a brief overview of the subsystem functionality. Specify the design of the subsystem. Specify the dependencies of the subsystem to other subsystems							
File Structure & Global Data: Provide any information relating to database requirements and the data that resides in the database.							
Requirement ID	Requirement Description	Type (e.g. Functional, privacy, security etc)	Technical Specifications for Development Team	Acceptance Test [AT]	Design Constraints	Modifications made to the Requirement and its source	Others
Interface Design: Discuss how the subsystem and its functionality interface with other subsystems, how the subsystem interfaces with external data. Provide brief overview of the design aspects of the subsystem. Include use-case diagrams when necessary.							

Requirements Traceability Matrix								
Requirement ID	Requirement Name	Source of Requirement	Requirements Document	Subsystem Number	Test Specs	Test Case(s)	Successful test Verification	Remarks

B.3 Requirements Workbook Guide

Project Name-Document Name: Project Team:
Version:
File Name:
Revision Date:

Document Author(s)

Project Sponsor:

Table of Contents

1. Overview

The Overview section consists of five subsections and provides for an executive level overview.

1.1 Purpose of this document
Describes the purpose of the document, and the intended audience.

1.2 Scope of this document
Describes the scope of the requirement specification. This section also details any constraints that were placed upon the requirement elicitation process, such as schedules, costs, or the software engineering environment used to develop requirements.

1.3 References
Identify sources of information used to develop this document, such as IEEE or template provided by instructors if any.

1.4 System Overview
Provides a brief overview of the component or system that is covered by the specification. This section should be brief, since it is included only to help the reader quickly understand what is being specified.

1.5 Business Context
Provides an overview of the business organization sponsoring the development of this product. This overview should include the business's mission statement and its organizational objectives or goals.

2. General Description

This section consists of six subsections of brief descriptions that provide understanding of the context for the proposed effort.

2.1 Product Functions
Describes the general functionality of the product, which will be discussed in more detail below.

2.2 Similar System Information
Describes the relationship of this product with any other products. Specifies if this product is intended to be stand-alone, or else used as a component of a larger product. If the latter, this section discusses the relationship of this product to the larger product.

2.3 User Characteristics
Describes the features of the user community, including their expected expertise with software systems and the application domain.

2.4 User Problem Statement
This section describes the essential problem(s) currently confronted by the user community.

2.5 User Objectives
This section describes the set of objectives and requirements for the system from the user's perspective. It may include a "wish list" of desirable characteristics, along with more feasible solutions that are in line with the business objectives.

2.6 General Constraints
Lists general constraints placed upon the design team, including speed requirements, industry protocols, and hardware platforms, and so forth.

3. Requirements

This section consists of twelve subsections. This section states the functions required of the software in quantitative and qualitative terms, and what the system must do to completely fulfill the owner/user′s expectations. The requirements should answer the following questions:

How are inputs transformed into outputs?
Who initiates and receives specific information?
What information must be available for each function to be performed?

Each paragraph (or group of paragraphs) should contain a reference identifying the source of the requirement. Each requirement (sentence or paragraph) should be numbered, using a numbering scheme that allows for inserting additional requirements later, e.g., FR-1.1, or A-1.1, etc. Only one requirement should be defined per numbered item.

Each requirement should be classified as one of the following:
Mandatory: Absolutely essential feature; project will be canceled if not included.
Required: Individual features are not essential, but together they affect the viability of the project.
Desired: Nice-to-have feature; one or more of these features could be omitted without affecting the project viability.

3.1 Goals
Provide a clear list of the expectations of a new system or function(s), both in terms of what must be improved and what must be retained from the current processes. All detailed requirements should address one or more of these goals.

3.2 Input and Output Requirements
Provide a description of all manual and automated input requirements for the software product such as data entry from source documents and data extracts from other applications, as well as all output requirements for the software product such as printed forms, reports, display screens, files and other work products the system will process and produce.

3.3 Data Requirements
Identify the data elements and logical data groupings that will be stored and processed by the software product. Include archiving data requirements and sensitivity of data.

This section is supported by a data model. An accompanying data dictionary should be included in an appendix.

3.4 Functional Requirements

Delineate, at a detailed level, computer system requirements within the context of the processes they must support. Each functional requirement should be specified in a format similar to the following and the listing should be based on the priority of the functional requirement.

	Description	Criticality	Technical issues	Cost and Schedule	Risks	Dependencies with other requirements	Others
FR-1							
FR-2							

Brief description of the variables documented in the FR is as follows:
Description - A full description of the requirement.
Criticality - Describes how essential this requirement is to the overall system.
Technical issues - Describes any design or implementation issues involved in satisfying this requirement.
Cost and schedule - Describes the relative or absolute costs associated with this issue.
Risks - Describes the circumstances under which this requirement might not able to be satisfied, and what actions can be taken to reduce the probability of this occurrence.
Dependencies with other requirements - Describes interactions with other requirements.
Others as appropriate, if any.

3.5 Performance Requirements
Portray owner/user-defined standards for system operations, relating to hours of operations, system response time, volumes, growth, and reliability.

3.6 Systems and Communication Requirements

Describe hardware and software interface requirements, as well as the connectivity and data interchange requirements in terms of types and volumes of data, location, and frequency of use.

3.7 System Security Requirements

Provide details of the security classification of the data handled by the system, special handling required for the data, and the types and levels of protection and control required for user access to the data. This section should also details telecommunications security aspects, e.g., , workstation/server, network, system, dial-up access, etc.

3.8 Back up and Recovery Requirements

Provide details of back up and recovery requirements. If software is identified as mission essential a continuity of operations plan must be developed.

3.9 Support Considerations

A description of any special or unusual support considerations that this system or component might require e.g., first time a UNIX system will be shipped to the field.

3.10 Hardware Requirements

3.10.1 Hardware Functionality

This section should cover the required capabilities of the hardware, e.g., requirement for the hardware to support multiple operating systems, or must support ethernet.

3.10.2 Hardware Characteristics

Required characteristics of the hardware. At a minimum this should include any requirements for diagnosis of the hardware.

3.11 Software Requirements

3.11.1 Software Functionality

This section should cover the required capabilities of the software, e.g. databases, operating systems, communications (remote access), diagnostics.

3.11.2 Software Characteristics

This section should cover the required characteristics of the software, e.g. reusability of code, packaging.

3.12 Usability Requirements

This section should define the requirements associated with ease of use, including menu structures, screen/window designs, screen colors, screen navigation, maximum input lines per screen, query capabilities, unattended installation, report layouts, online help and other interfaces to users and/or supervisors.

4. Design Requirements

This section consists of three subsections and details the technical requirements.

4.1 Data Flow and Software Structure

Describe the important data flow paths of your design and provide a diagram to show the flow of data. Show another diagram to depict the architectural flow of the system. State the relationships between the subsystems.

4.2 Subsystem Design

Specify the design and functions for each subsystem and module in your software system. Feel free to use diagrams in this section to help describe the subsystems/modules.

4.3 Design Constraints

Document any design constraints that should be taken into consideration during the system design phase.

5. Requirements Traceability Matrix

5.1 Description of Matrix Fields

Develop a matrix to trace the requirements back to the project objectives identified in the Project Plan and forward through the remainder of the project life cycle stages. Place a copy of the matrix in the Project File. Expand the matrix in each stage to show traceability of work products to the requirements and vice versa. The requirements traceability matrix should contain the following fields:

- A unique identification number containing the general category of the requirement (e.g., SYSADM) and a number assigned in ascending order (e.g., 1.0; 1.1; 1.2).
- The requirement statement.
- Requirement source (Conference; Configuration Control Board; Task Assignment, etc.).
- Software Requirements Specification/Functional Requirements Document paragraph number containing the requirement.
- Design Specification paragraph number containing the requirement.
- Subsystem containing the requirement.
- Test Specification containing the requirement test.
- Test Case number(s) where requirement is to be tested (optional).
- Verification of successful testing of requirements.
- Modification field. If requirement was changed, eliminated, or replaced, indicate disposition and authority for modification.
- Remarks.

5.2 Requirements Traceability Matrix

See example:
Project Name
Requirements Traceability Matrix

Unique Number	Requirem ent Name	Source of Requirement	Software Reqs. Document	Design Spec.	Program Module	Test Spec.	Test Case(s)	Successful Test Verification	Modification of Requirement	Remar ks
Objective 1:										

6. Glossary

A glossary of terms and definitions used in the Requirements Specification that might not be known to the reader or open to misinterpretation. If a standard glossary is available this might be referenced in the reference section and included with the specification to any readers or reviewers of the specification.

7. Document Revision History

Appendix C Surveys

Appendix C.1 Student Survey

We appreciate your feedback on the process model used for this class. All answers that you provide for the survey will be kept confidential and will not affect your grade in any way. Therefore, honest feedback will be helpful and appreciated.
What are the first four digits of your student id?

Background Information

1. What is your major?
a. Computer science b. Management c. other___________________

2. What course are you taking now?
a. CSC 326 b. E-Commerce Practicum

3. Do you have prior work experience with software engineering?
a. Less than one b. 2-5 c. Over five years d. None

4. What roles did you fulfill for your team? (Circle all that apply for Qs 4 and 5)
a. Programmer b. SQA c. Technical lead d. Project Manager e. Technical Writer

5. Please indicate all the techniques you have used to determine the requirements for your project / system?
a. Phone interview b. Email c. Whiteboards d. Scenario Analysis e. Face to Face

Please check one for each statement

Please provide feedback on your group and on the process model, in general.	**Strongly Agree**	**A**	**I**	**D**	**Strongly Disagree**
Overall, my group was successful in completing the project.					
My group had adequate interaction with the customer.					
I have a clear understanding of ADaPT.					
Software process model is essential for developing system.					
I am familiar with evolutionary process model.					
I would characterize ADaPT as lightweight in comparison to evolutionary model.					
In comparison to other process models, ADaPT reduces the planning and documentation effort.					
Please provide feedback on the various templates and process documents.	**SA**	**A**	**I**	**D**	**SD**
The project plan workbook template was easy to read and use.					
The project plan workbook template provided guidance for project planning.					
The project plan template was of appropriate length.					
The requirements template was easy to read and use.					
The requirements template was of appropriate length.					
The requirements document guide provided guidance for documenting requirements.					
My group produced a ***quality*** requirements document.					
We were able to effectively incorporate emerging new requirements in the system.					
ADaPT handles evolving system requirements effectively.					
My group's project required the production of a security policy.					
My group's project required the production of a privacy policy.					
Please provide feedback on the risk analysis meeting process.	**SA**	**A**	**I**	**D**	**SD**
My group held at least one risk analysis meeting.					
I have a clear understanding of the system requirements.					
The system requirements changed during the project lifecycle.					

The risk analysis meetings enabled us to address changing system requirements.					
Please provide feedback on the risk analysis meeting process.	**SA**	**A**	**I**	**D**	**SD**
The risk analysis meeting helped to identify key risks threatening the project.					
During the risk analysis meeting: 1. My group identified conflicting requirements and design constraints.					
2. My group resolved conflicts upon their identification.					
3. My group looked for conflicts among security and privacy statements.					
4. My group explicitly identified conflicting policy statements.					
5. My group resolved policy statements conflicts.					
6. My group looked for conflicts between requirements and policy statements.					
7. My group resolved identified conflicts between requirements and policy statements.					
8. My group resolved conflicts between requirements and policy statements.					
Please provide feedback on the ADaPT Planning and Design Phase.	**SA**	**A**	**I**	**D**	**SD**
Gathering requirements as scenarios was helpful.					
Scenario analysis conducted to operationalize requirements was useful.					
Goal and Scenario analysis helped us understand the system better.					
Clustering of related goals into subsystems was helpful in implementation planning.					
The planning phase enabled us to start the project with well-defined requirements.					
Our group devoted more time to discussing design issues than to documentation.					
Using Whiteboards for design discussions is effective.					
Constructing a high-level architecture model was sufficient for design documentation.					
A separate design document should be maintained instead of having whiteboard copies.					
The planning phase activities were performed in less than three weeks.					
Each subsystem development cycle lasted for roughly three weeks.					
Our group monitored the time spent implementing each subsystem.					
Our group monitored the tasks completed in each cycle.					
Our group accomplished equally in all the cycles.					
Please provide feedback on the ADaPT Implementation and Testing Phase.	**SA**	**A**	**I**	**D**	**SD**
Our group used Pair Programming to implement system.					
If yes, pair programming was effective.					
Acceptance tests for a subsystem were written prior to implementation.					
Writing acceptance tests before implementation improves system-quality.					
Unit tests were performed often.					
Having an SQA team enabled us to test the system effectively.					
SQA team performed integration of subsystems.					
Integrating subsystems was not a difficult task.					
System testing was performed often.					
Customer validation was performed at the end of every cycle.					
The Customer provided feedback on the evolving system at the end of every cycle.					
Our customer is satisfied with the validated system thus far.					
Please provide feedback on the Project outcome.	**SA**	**A**	**I**	**D**	**SD**
I was confident about the product release.					
Our project is a success.					
Initial planning and documentation effort was minimal.					
ADaPT enabled us to develop a high quality system.					
Our project will be delivered on time.					

Appendix C.2 Instructor Survey

I appreciate your feedback on the process model.

Please check one for each statement

Statement

Please provide feedback on group-*X*:	**Strongly Agree**	**A**	**I**	**D**	**Strongly Disagree**
The group was successful in completing their project.					
The group met its course deliverables.					
The customer is satisfied with the group’s progress.					
The group developed a high-quality system.					
The system meets customer’s requirements.					
The communication amongst team members was good.					
The group was very organized.					
The cooperation amongst team members was good.					
Customer feedback to the students was timely and frequent.					

Appendix C.3 Customer Survey

I appreciate your feedback on the team project.

Please check one for each statement

Indicate the team's performance that developed your system

The members of the team:	**Strongly Agree**	**Agree**	**Indifferent**	**Disagree**	**Strongly Disagree**
• Successfully completed the project.					
• Delivered the system on time.					
• Incorporated all requirements in the delivered system.					
• Developed a high quality system.					
• Produced useful and essential artifacts.					
• Communicated well amongst team members.					
• Demonstrated excellent organization skills.					
• Cooperated well amongst team members.					
• Provided progress reports on the project.					

Additional Comments

Appendix D Project Descriptions

D.1 Group Project Descriptions for E-Commerce Practicum

Groups	Team Members in the group	Sponsor	Project Description	Project Constraints	Risk Analysis	System Features	Platform and Languages Used
Art Museum Portal	The team was comprised of five graduate students: two computer science majors and three management majors.	Hickory Museum of Art	To develop an interactive website to allow museum members to register for classes and activities, make donations, and purchase merchandise from the site.	The sponsors' proprietary DB had to be integrated with the front-end developed by the team. The website needed to address the needs of members with varying backgrounds, which was challenging.	The project scope was hard to determine since the customer gave the team complete authority to define the scope.	The final system provided the following features: ability to store member information, ability to maintain security of data, form creation, and API to enable user to update information.	Windows NT, HTML, SQL, JavaScript
Conference Registration System	The team was comprised of five graduate students: two computer science majors and three management majors.	NCSU	To develop an interactive web-based conference registration system to allow the individuals to securely register for the 2nd SREIS conference.	The e-studio did not have all needed software to complete the project. The performance measures envisioned by the customer were beyond the control of developers e.g. bandwidth or modem type used by customer.	The team lacked highly skilled programmers; the project needed skilled programmers to implement some of the system requirements.	The final system provided the following features: ability to store registrant information, SSL implementation, and email confirmation to registrants with details of registration.	Windows NT, HTML, SQL Server, MS Access, ASP, Visual Interdev, IIS Server.

Groups	Team Members in the group	Sponsor	Project Description	Project Constraints	Risk Analysis	System Features	Platform and Languages Used
Lipsinc and Centra Web Training System	The team was comprised of six graduate students: three computer science majors and three management majors.	Lipsinc and Centra	To develop an interactive web training experience by merging the technologies provided by Lipsinc and Centra. Lipsincs' software application facilitates creation of realistic facial animation for digital characters using audio input. Centra is a leader in providing software application and services in e-learning.	The team faced Non Disclosure Agreement issues that prohibited the companies from sharing proprietary software. This prevented the team from incorporating most of the system requirements.	The team faced two challenges: integrating two different software applications and understanding the code implemented by other developers.	The final system provided the following features: ability to integrate text with the speech engine, and ability to display of 3D characters in a separate window.	Windows NT, HTML, Power point, Visual C++.
Wireless Enabled Online Banking System	The team was comprised of four graduate students: two computer science majors and two management majors.	IBM	To develop an online banking system using XForms and to acquire data from existing web services in XML format. In this project, a remote browser-based client initiates a request for data to a server side XForms application. The application acquires the desired web services data, manipulates data, renders data required by the XForms processor, and delivers the data back to the client. The project also involves implementing this procedure with a wireless-based client.	The team was responsible for developing a complicated system in a 15-week period. The project scope was not well defined. The team members lacked experience with XForms and had to overcome a learning curve due to their inexperience with XForms.	The specification for the pre-release X-forms 1.0 version was not clear because of insufficient customer feedback.	The final system incorporated the following features: ability to access account and user information, ability to maintain information security of data, and API to enable conversion of application data to wireless format and vice versa.	Windows NT, tomcat servlet, apache web server, X-Smiles browser, web sphere 4.0 server.

D.2 Group Project Descriptions for Software Engineering Course

Groups	Team Members in the group	Sponsor	Project Description	Project Constraints	Risk Analysis	System Features	Platform and Languages Used
Group 1- Faculty Web Publications System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based system to deposit professors' publication information, and view other professors' publications and research grant information. This would allow the professors to gain familiarity with their colleagues' research areas.	The website needed to address the needs of faculty members with varying backgrounds and requirements, which was challenging. Additionally, the system had to comply with the NCSU CSC department's web site style / look and feel.	The team members had varying programming experience. The project needed skilled programmers to accomplish certain features of the system requirements. The students faced several course requirements- e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store professors' publications, API to allow professors to add, remove, update publication and grants information, and export files to latex.	Windows NT, Java Beans, My SQL, JSP, Apache server.
Group 2- Faculty Web Publications System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based system to deposit professors' publication information, and view other professors' publications and research grant information. This would allow the professors to gain familiarity with their colleagues' research areas.	The website needed to address the needs of faculty members with varying backgrounds and requirements, which was challenging. Additionally, the system had to comply with the NCSU CSC department's web site style / look and feel.	The team members had varying programming experience. The project needed skilled programmers to accomplish some of the system requirements. The students faced several course requirements- e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store professors' publications, API to allow professors to add, remove, update publication and grants information, and export files to latex.	Windows NT, Java Beans, My SQL, JSP, Apache server.

Groups	Team Members in the group	Sponsor	Project Description	Project Constraints	Risk Analysis	System Features	Platform and Languages Used
Group 3- Faculty Web Publications System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based system to deposit professors' publication information, and view other professors' publications and research grant information. This would allow the professors to gain familiarity with their colleagues' research areas.	The website needed to address the needs of faculty members with varying backgrounds and requirements, which was challenging. Additionally, the system had to comply with the NCSU CSC department's web site style / look and feel.	The team members had varying programming experience. The project needed skilled programmers to accomplish some of the system requirements. The students faced several course requirements- e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store professors' publications, API to allow professors to add, remove, update publication and grants information, and export files to latex.	Windows NT, Java Beans, My SQL, JSP, Apache server.
Group 4- Conference Registration System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based conference registration system to allow the individuals to securely register for the 2nd SREIS conference.	The computer labs did not have all needed software to complete the project. The performance measures envisioned by the customer were beyond the control of developers e.g., bandwidth and modem type used by the user.	The team members had varying programming experience. The project needed skilled programmers to implement some of the system requirements. The students faced several course requirements- e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store registrant information, and email confirmation to registrants with details of registration.	Windows NT, HTML, SQL Server, MS Access, ASP, Visual Interdev, Java beans, JSP, J2EE Apache Server.

Groups	Team Members in the group	Sponsor	Project Description	Project Constraints	Risk Analysis	System Features	Platform and Languages Used
Group 5-Conference Registration System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based conference registration system to allow the individuals to securely register for the 2nd SREIS conference.	The computer labs did not have all needed software to complete the project. The performance measures envisioned by the customer were beyond the control of developers e.g., bandwidth and modem type used by the user.	The team members had varying programming experience. The project needed skilled programmers to implement some of the system requirements. The students faced several course requirements-e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store registrant information, and email confirmation to registrants with details of registration.	Windows NT, HTML, SQL Server, MS Access, ASP, Visual Interdev, Java beans, XML, JSP, J2EE Apache Server.
Group 6-Conference Registration System	The team was comprised of six senior computer science students.	NCSU	To develop an interactive web-based conference registration system to allow the individuals to securely register for the 2nd SREIS conference.	The computer labs did not have all needed software to complete the project. The performance measures envisioned by the customer were beyond the control of developers e.g., bandwidth and modem type used by the user.	The team members had varying programming experience. The project needed skilled programmers to implement some of the system requirements. The students faced several course requirements-e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store registrant information, and email confirmation to registrants with details of registration.	Windows NT, HTML, SQL Server, MS Access, ASP, Visual Interdev, Java beans, JSP, J2EE Apache Server.

Groups	Team Members in the group	Sponsor	Project Description	Project Constraints	Risk Analysis	System Features	Platform and Languages Used
Group 7-Conference Registration System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based conference registration system to allow the individuals to securely register for the 2nd SREIS conference.	The computer labs did not have all needed software to complete the project. The performance measures envisioned by the customer were beyond the control of developers e.g., bandwidth and modem type used by the user.	The team members had varying programming experience. The project needed skilled programmers to implement some of the system requirements. The students faced several course requirements- e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store registrant information, and email confirmation to registrants with details of registration.	Windows NT, HTML, My SQL, RSA, Triple DES, Java scripts, CSS, IE 5.5plus, PHP CGI, Apache Server.
Group 8-Conference Registration System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based conference registration system to allow the individuals to securely register for the 2nd SREIS conference.	The computer labs did not have all needed software to complete the project. The performance measures envisioned by the customer were beyond the control of developers e.g., bandwidth and modem type used by the user.	The team members had varying programming experience. The project needed skilled programmers to implement some of the system requirements. The students faced several course requirements- e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store registrant information, and email confirmation to registrants with details of registration.	Windows NT, HTML, SQL Server, MS Access, ASP, Visual Interdev, Java beans, JSP, J2EE Apache Server.

Groups	Team Members in the group	Sponsor	Project Description	Project Constraints	Risk Analysis	System Features	Platform and Languages Used
Group 9- Conference Registration System	The team was comprised of seven senior computer science students.	NCSU	To develop an interactive web-based conference registration system to allow the individuals to securely register for the 2nd SREIS conference.	The computer labs did not have all needed software to complete the project. The performance measures envisioned by the customer were beyond the control of developers e.g., bandwidth and modem type used by the user.	The team members had varying programming experience. The project needed skilled programmers to implement some of the system requirements. The students faced several course requirements- e.g., homework and exams, along with the challenge of developing a high quality system.	The final system incorporated the following features: ability to store registrant information, and email confirmation to registrants with details of registration.	Windows NT, HTML, SQL Server, MS Access, ASP, Visual Interdev, Apache Server.