Submitted to: 10th Anniversary IEEE Joint Requirements Engineering Conference (RE'02)

Towards Improved Requirements Practices in Agile Software Development

Hema L. Srikanth and Annie I. Antón

College of Engineering
North Carolina State University
Raleigh, NC, 27695-7534 USA
{hlsrikan, aianton}@eos.ncsu.edu

Abstract

Traditional software processes, while rich with support for requirements activities, are not especially well suited for rapid software development. Agile software processes have recently received increased attention due to the need to develop software for rapidly changing environments; however they fail to properly support essential requirements engineering practices. This paper proposes a agile approach to software development, ADaPT (Agile 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 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 Introduction

In today's fast-paced and competitive commercial software development environment, speed and flexibility are essential. Companies are compelled to try newer ways to develop software, moving away from traditional approaches including the Waterfall and Spiral [Boe88] models [Jal00], which are not as viable in today's market driven environment. Many projects are cancelled or fail to meet customer expectations; studies have found that about two-thirds of all projects substantially overrun their estimates [SB01]. Traditional software process models are suitable for low-risk predictable projects. However, they are not considered flexible enough for complex projects with changing requirements [Bea99]. Traditional software development methodologies and the Capability Maturity Model (CMM) [PCC93] impose a disciplined process, making an organization's software process and practices more predictable. However, these methodologies are bureaucratic, increase software costs due to excessive documentation, and reduce development speed [DBC88, Fow00]. More recently, evolutionary development has sought to "evolve" or "grow" some or all a system's functionality into a final product [CSW97]. Evolutionary processes are iterative in nature, basically incorporating mini-waterfalls within each development cycle [Hig98].

The introduction of each new process model has been marked by a slight decrease in the amount of required documentation. However, planning consistently requires substantial resources (time and effort), impacting delivery speeds. Agile methodologies shift the focus away from project documentation to techniques used to develop/write source code. Agile prototyping, which provides the basis for this work, helps accelerate development speed by reducing excessive planning and documentation [Hig98]; it is adaptive in nature, emphasizing adapting lessons learnt during every cycle instead of predicting the minuscule elements of the project details [Hig98].

In this paper 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 agile prototyping as a viable approach to rapid software development and introduce our Agile Development and Prototyping Technique (ADaPT) to provide flexibility, adaptability and increased productivity in software development.

ADaPT enables software teams to produce software incrementally; ensures that developers work together effectively; and enables complex products to be developed efficiently. ADaPT provides additional benefits including: increased customer involvement to clarify requirements; improved ability to handle risk and uncertainty; better quality software; and increased development speed.

Researchers in the North Carolina State University (NCSU) Requirements Engineering Lab have been addressing the need for requirements rigor during rapid software development [ACS01, ACE01]. Our earlier work resulted in the development of the EPRAM (Evolutionary Prototyping and Risk Analysis and Mitigation) model [CAD01], which was employed during nine Web-based electronic commerce (e-commerce) development efforts.

While the model proved successful, it was still perceived to be somewhat heavy and the developers yearned for a more lightweight process. We attribute this "heaviness" to the emphasis placed on the EPRAM's adherence to the CMM [Jal00], which inevitably makes it challenging to remove much of the "bureaucracy", whether perceived or real, in software development efforts. ADaPT builds upon the lessons learned in applying the EPRAM model over the course of one calendar year by addressing the risks and challenges inherent in small, rapid prototyping projects. For example, e-commerce software developers are often under pressure to develop software at a 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. Similar to the EPRAM model, ADaPT adheres to the spirit of CMM as prescribed in [ACS01], which provides a tailored CMM for small e-commerce development teams. The ADaPT is currently undergoing validation in various e-commerce development efforts; our preliminary findings suggest that ADaPT works effectively for rapid software development.

The remainder of this paper is organized as follows. Section 2 describes the relevant related work. ADaPT is introduced in Section 3, using examples to demonstrate scenario and goal-based requirements analysis to elicit and structure requirements into logical subsystems. Section 4 provides a brief operational example of how scenarios and goals are structured into subsystems in ADaPT. Finally, Section 5 provides a summary and discusses our plans for future work.

2 Background and Related Work

This section briefly summarizes prominent agile methodologies, focusing on their requirements practices, and discusses the relevant work in goal and scenario driven requirements engineering (RE) and risk management.

2.1 Agile Methodologies

Most software processes and development efforts are chaotic, often characterized as "code and fix". The software is written without proper planning, and the system design efforts are often ad-hoc. In chaotic development, bugs are discovered increasingly in the latter lifecycle stages, and therefore the product is not released on schedule. Traditional, more disciplined and predictable approaches emphasize document maintenance and project planning. However, these methodologies are often bureaucratic, resulting in heavy overhead costs and reduced delivery speed. In response to the inevitable overhead associated with traditional methodologies, software practitioners have eagerly adopted the "Agile Methodologies" (or "Lightweight Methodologies") [Hig01].

Agile methodologies primarily focus on the techniques to develop/write source code instead of earlier software development activities, such as requirements and design. Some believe that undue documentation reduces flexibility and delivery speed [Hig98]. Although the approach proposed in this paper, calls for requirements to be documented, the process remains flexible by structuring the RE activities using proven goal and scenario based analysis and specification techniques.

Fowler distinguishes agile methodologies from heavier processes as follows: they are adaptive rather than predictive and they are people-oriented rather than process-oriented [Fow00]. Agile software development advocates profess these methods are superior for providing customers with what they want, when they want it, and with acceptable defect rates. However, to date, there have been no quantifiable studies to support or refute these claim; our validation efforts aim to provide aggregate results. We now summarize several common agile methodologies, examined to develop the ADaPT.

2.1.1 Extreme Programming

Extreme Programming (XP) has proven to work well for small, risky projects with dynamic requirements [Bec00]. The techniques for gathering requirements in XP are a radical departure from that of more traditional approaches. In XP, customer requirements are written in natural language, on informal "User Story" cards (quite similar to use cases [Jac92]). These User Story cards are never formalized; no relationships or dependencies between the cards appear to be identified or documented. Software developers place time estimates and customers assign priorities to each card. Together, developers and customers play a "Planning Game" in which the customer chooses those User Stories that comprise the most important content for a short, incremental deliverable that normally takes about one month to complete. XP developers intentionally work under the assumption that any remaining requirements may never be integrated into the product and therefore no infrastructure is built in for possible future requirements. Instead, development is focused on the content of short, customer-prescribed increments.

As in contextual design [BH98], the customers must be readily available and accessible to the developers for clarifying and validating requirements throughout the implementation process. Each short implementation increment is accepted and validated by the customer. At which point, the remaining User Stories are re-examined for possible requirement and/or priority changes and the Planning Game is re-played for the next implementation increment.

2.1.2 Scrum

Scrum is used on small projects: the development teams are comprised of four to seven individuals and the projects are divided into several iterations [SB01]. In

Scrum's planning phase, developers elaborate the architecture of the system. The system is incrementally developed in short development cycles called "sprints," and each sprint typically lasts no more than a month. Before each sprint, the system requirements are listed as a "backlog" [RJ00]; each backlog represents a prioritized sequence of tasks to be accomplished in a given sprint. Scrum meetings are held often to track progress, plan the next sprint, and analyze any obstacles. The planning phase is short since initial assumptions change at the end of every sprint as the system evolves. Unlike traditional models the complete requirements for a project are not written up-front since the customer is presumably indecisive at the beginning of the project.

2.1.3 Crystal

In Cockburn's "Crystal" [Coc99], software development is characterized as a co-operative "game" where all team members work towards achieving a common goal. The team members use "markers" to track "moves"; markers are similar to project status indicators in that they inform and remind team members at each point in the software development game. A successful game ends with an operating system; the game "residue" are the markers that inform and assist the players of the next game. In Crystal, requirements are elicited during "question and answer" sessions and the contents of this session are written in requirements document.

2.1.4 Adaptive Software Development

In Adaptive Software Development (ASD), the traditional and static "Plan-Design-Build" approach is replaced with a "Speculate-Collaborate-Learn" approach [Hig98]. Planning is viewed as a paradox in ASD, since outcomes are often unpredictable. In traditional software process models, deviations from plans are considered mistakes that should be corrected. In contrast, deviations in ASD guide developers to achieve the correct solution [Fow01]. ASD developers must collaborate effectively to address 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 each development cycle's results to adapt to the next [Fow01, Hig98]. In ASD, requirements are elaborated and documented for the components developed during a given cycle. The progress of individual components is tracked and documented in a requirements document [Hig98].

2.2 Goal and Scenario Driven Analysis

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. Goal-driven RE approaches focus on why systems are constructed, providing the motivation and rationale to justify software requirements. 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 the 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) 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. The 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 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.

2.3 Managing Risks During RE

Risk management during software development is essential for project success, yet challenging in rapid development environments. Boehm's Spiral Model [Boe88] explicitly supports risk management and his Win-Win Model [BI96] relies on risk analysis techniques to manage uncertainty. In ASD, planning for subsequent cycles are driven by risk analysis [Hig00]. The EPRAM model [CAD01] incorporates the risk and compliance assessment activities of [AE01a] and the risk identification and ranking processes of Baskerville *et al.* [BS96] to form a comprehensive risk mitigation strategy.

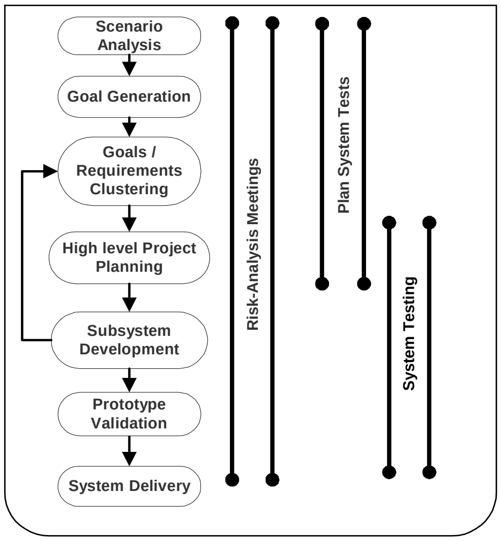
The ADaPT borrows from the EPRAM model in that risks are identified, analyzed and managed before and during every cycle. ADaPT ensures the compliance of the documented system requirements with established security and privacy policy [AE01a] via risk analysis meetings. Risk-analysis meetings are used to formulate a comprehensive risk mitigation strategy at the beginning of the project and these risks are subsequently evaluated during every risk analysis meeting.

3 ADaPT

Two main phases comprise of 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 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 a agile process. Documentation in maintained in spreadsheets[1]; 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 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.

Figure 1: High Level Overview of ADaPT



3.1 Planning and Design Phase

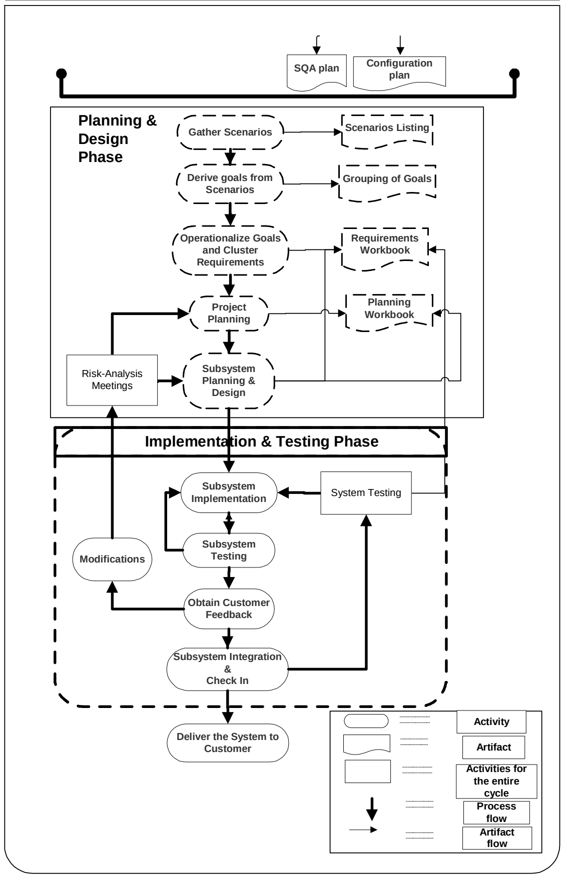
During the planning and design phase, requirements are elicited in the form of scenarios and elaborated with goals before the development team begins actual project planning. Two artifacts are produced during this phase: a *planning workbook* and a *requirements workbook*. The requirements workbook documents 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

Figure 2: ADaPT Process Model



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

[1] The planning workbook spreadsheet and requirements document workbook templates are available at: http://ecommerce.ncsu.edu/studio/resources.html.

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

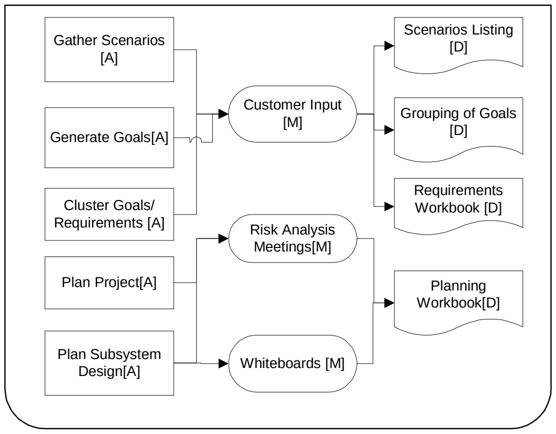
G_1: `(System) PROMPT user to enter search keywords`
G_2: `(System) SEARCH for keyword matches`
G_3: `(System) GENERATE search results web page`
G_4: `(System) DISPLAY search results web page`

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.

Figure 3: ADaPT Planning and Design Phase



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 subsytem 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].

Activity 4: Plan Project

Once logical requirements are organized into subsystems, the subsystem with the highest priority (as prescribed by the customer) 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.

3.2 Implementation and Testing Phase

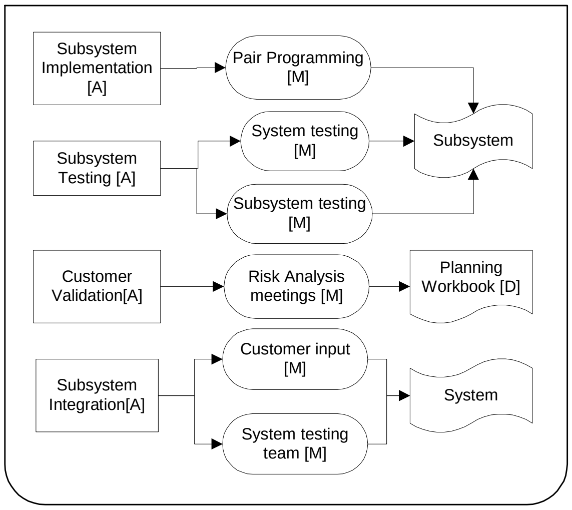
This section describes the ADaPT implementation and testing phase. Every subsystem is developed according to Figure 2. Figure 4 graphically depicts the four key activities that comprise the subsystem implementation and testing phase.

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 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.

Figure 4: ADaPT Implementation and Testing Phase



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 are 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.

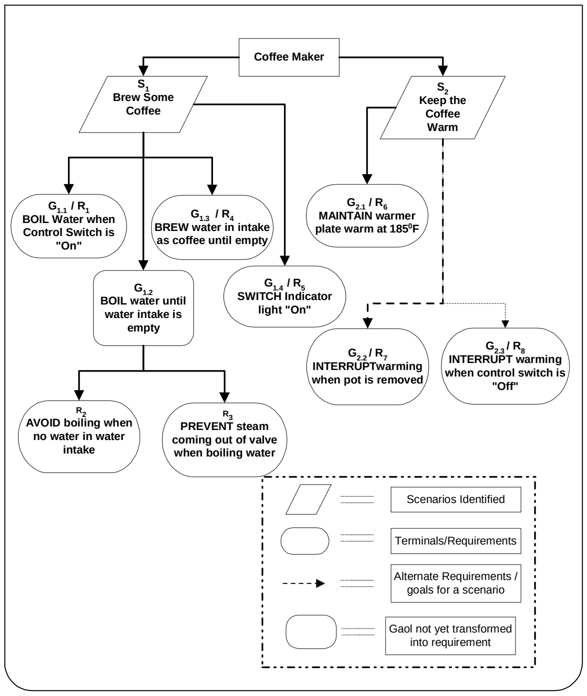
4 Operational Example

In this section, we demonstrate how ADaPT is employed to design an ordinary coffee maker, with simple functionality (see Figure 5). For convenience and simplification, we elaborate only the system viewpoint; therefore goals are not expressed with the actor information since there is only one viewpoint. Scenarios are identified to reason about the envisioned functionality for the coffee maker and elaborated with goals using inquiry. The terminal nodes in Figure 5 represent the system requirements (shown as R_x). After the requirements are generated, they are grouped into subsystems. The requirements for coffee maker are grouped into three subsystems.

SS_1: `Water Boiler:` R_1, R_2, R_3
SS_2: `Coffee Brewer:` R_4, R_5.
SS_3: `Coffee Warmer:` R_6, R_7, R_8.

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?"*

Figure 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"
G1.3 / R4 BREW water in intake as coffee until empty
G2.1 / R6 MAINTAIN warmer plate warm at 185⁰F
G1.2 BOIL water until water intake is empty
G1.4 / R5 SWITCH Indicator light "On"
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

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 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."

The two scenarios were elaborated with goals; these goals were allocated to three subsystems. In XP, these scenarios are listed as "user stories", which would require two development cycles. However, by grouping related requirements to form subsystems in ADaPT, developers will avoid redundancy and the implementation process will be more efficient. We now summarize our experiences to date with ADaPT and discuss our current validation efforts.

5 Summary and Future Work

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 the 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 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.

In summary, ADaPT is based upon four solid elements: a firm CMM basis for maturity; inclusion of proven RE practices; an agile prototyping to accommodate flexibility and speed; 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 agile software development.

Case studies are serving as the primary mechanism for validating the ADaPT. Validation is currently underway via its use in several e-commerce development projects for IBM, NCSU, and a regional art museum within the NCSU Electronic Commerce Studio[2] -- a joint venture between the Computer Science department and the College of Management [AE01b]. Early indications 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,

[2] http://www.ecommerce.ncsu.edu/studio/

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 immediate plans involve further 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.

Acknowledgements

This work was partially funded by the NCSU E-Commerce Learning and Asea Brown Boveri. The authors wish to thank Ryan Carter, Aldo Dagnino, Julie Earp, Michael Rappa and Laurie Williams for discussions during development of the ADaPT.

References

[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.

[ACD01] A.I. Antón, R.A. Carter, A. Dagnino, J.H. Dempster and D.F. Siege. Deriving Goals from a Use-Case Based Requirements Specification, *Requirements Engineering Journal*, Vol. 6, pp. 63-73, May 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.

[ACE01] A. I. Antón, R. A. Carter, J. B. Earp and L. A. Williams. EPRAM: Evolutionary Prototyping Risk Analysis & Mitigation, NCSU Technical Report, TR-2001-08, Auust 20, 2001.

[AE01a] A.I. Antón and J.B. Earp. Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems. in *E-Commerce Security and Privacy*, ed. by A.K. Ghosh, Kluwer Academic Publishers, pp. 29-46, 2001.

[AE01b] A.I. Antón and J.B. Earp. The NCSU E-Commerce Studio: Supporting Multidisciplinary Project Driven Development for Secure Systems, Submitted to *Communications of the ACM*, 13 July 2001.

[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.

[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.

[BH98] K. Hotzblat and H. Beyer. Contextual Design., Morgan Kaufmann, San Francisco, CA, 1998.

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

[Boe81] B. Boehm. *Software Engineering Economics*, Prentice Hall, 1981.

[Boe88] B. Boehm. A Spiral Model for Software Development and Enhancement, *IEEE Computer*, 21(5), pp. 61-72, 1988.

[BS96] R.L. Baskerville & J. Stage. Controlling Prototype Development Through Risk Analysis, *MIS Quarterly*, 20(4), pp. 481-504, December 1996.

[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*, Toronto, Canada, pp. 94-101, August 2001.

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

[CW00] A. Cockburn and L. Williams. The Costs and Benefits of Pair Programming, Extreme Programming Explained, Addison Wesley, pp. 223-243, *XP 2000*.

[CSW97] Cunningham. D, Subrahmanian. E, and Westerberg. A. User-Centered Evolutionary Software Development Using Python and Java, *6th Int'l Python Conf.,* Carnegie Mellon University, PA, 1997.

[DBC88] Davis. A, Bersoff. E, and Comer. E. A strategy for comparing alternative software development life cycle models, *IEEE Transactions on Software Engineering*, 14(10), pp. 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.

[End75] A. Endres. An Analysis of Errors and Their Causes in System Programs, *IEEE Transactions on Software Engineering*, 1(2), pp. 140-149, June 1975.

[Fow00] M. Fowler, Put Your Process on a Diet, *Software Development*, Vol. 8, 2000, pp. 32-36.

[Fow01] M. Fowler. The New Methodology, ThoughtWorks Inc., November 2001.

[Hig98] J. Highsmith. Application Development Strategies, Cutter Information Corporation, August 1998.

[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.

[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

[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.

[LDM95] A. van. Lamsweerde, R. Darimont & P. Massonet. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt, *2nd Intl Symposium on Requirements Engineering (Re'95)*, York, UK, pp. 194-203, March 1995.

[Lev86] N. Leveson.. Software Safety: Why, What, and How, *Computing Surveys*, 18(2), pp. 125-163, June 1986.

[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.

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

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

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

[SB01] K. Schwaber & M. Beedle. *Agile Software Development with Scrum*, Prentice Hall, 2001.

[WPJ98] K. Weidenhaupt, K. Pohl, M. Jarke & P. Haumer. Scenarios in System Development: Current Practice, *IEEE Software*, 15(2), March/April 1998.

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