doi:10.1111/j.1365-2575.2007.00259.x

Agile requirements engineering practices and challenges: an empirical study

Balasubramaniam Ramesh,* Lan Cao† & Richard Baskerville‡

*Computer Information Systems Department, Georgia State University, 35 Broad St. NW, Atlanta, GA 30302-4015, USA, email: bramesh@gsu.edu, ‡email: baskerville@gsu.edu, and †Department of Information Technology and Decision Sciences, College of Business and Public Administration, Old Dominion University, Norfolk, VA 23529, USA, email: lcao@odu.edu

Abstract. *This paper describes empirical research into agile requirements engineering (RE) practices. Based on an analysis of data collected in 16 US software development organizations, we identify six agile practices. We also identify seven challenges that are created by the use of these practices. We further analyse how this collection of practices helps mitigate some, while exacerbating other risks in RE. We provide a framework for evaluating the impact and appropriateness of agile RE practices by relating them to RE risks. Two risks that are intractable by agile RE practices emerge from the analysis. First, problems with customer inability and a lack of concurrence among customers significantly impact agile development. Second, risks associated with the neglecting non-functional requirements such as security and scalability are a serious concern. Developers should carefully evaluate the risk factors in their project environment to understand whether the benefits of agile RE practices outweigh the costs imposed by the challenges.*

Keywords: requirements engineering, agile software development, requirements engineering risks, agile practices

INTRODUCTION

The traditional way of requirements engineering (RE) has been challenged by the rapidly changing business environment in which most organizations operate today. Most software development organizations are forced to deal with requirements that tend to evolve very quickly and become obsolete even before project completion (Boehm, 2000). Rapid changes in competitive threats, stakeholder preferences, software technology and time-to-market pressures severely challenge the development of systems based on pre-specified requirements (Boehm, 2000). A group of agile methods that seek to address this changed context has gained much attention in practice (Agile Alliance, 2001) and in the literature (Boehm, 2003). These methods include practices such as short iterations, frequent releases, simple design, peer reviews and the use of onsite customer participation. Agile methods support shorter development cycles in order to respond to complex, fast-moving and competitive marketplaces. For example, many agile methods advocate moving into coding without a centralized requirements analysis[1] and design phase. The requirements of the system emerge throughout the development process, while heavily relying on feedback from the customer. Evolving requirements, rapidly changing technology and time constraints cause the RE process for agile software development to be different (Cockburn, 2002; Levine *et al*., 2002).

As observed in recent studies (Erickson *et al*., 2005), there is a lack of empirical research on agile software development beyond a discussion of experience reports. Particularly, much of the work is focused on validating the applicability and usefulness of practices in eXtreme Programming (XP) (Beck *et al*., 1999), a popular agile development method. Beyond a few high visibility case studies (such as Microsoft® and Netscape®) (Cusumano & Yoffie, 2000) and anecdotal evidence, the current literature does not provide much insight on actual practices used in agile projects (Erickson *et al*., 2005) and their effectiveness in supporting RE activities. In this research, we investigate these issues based on a field study of 16 software development organizations.

RELATED RESEARCH

The important role of RE in ensuring successful systems development and minimizing project risks is well established (Orr, 2004). Two streams of research in agile software development (Erickson *et al*., 2005) consist of a small number of case studies and experience reports on adopting agile methods and empirical studies focused on pair programming. However, the literature that specifically addresses RE in agile development is still nascent. Actually, the term ‘requirements engineering’ is avoided in the agile community as it is often taken to imply heavy documentation with significant overhead.

Although some researchers argue that agile software development approaches simply repackage established techniques (Merisalo-Rantanen *et al*., 2005), RE in an agile environment has been recognized as different in other studies (e.g. Paetsch & Maurer, 2003). RE includes activities to identify, analyse, document and validate requirements for the system to be developed. Agile methods such as XP conducts RE throughout the development life cycle in small, informal stages (Beck *et al*., 1999). The customer is onsite as a member of the development team and participates in writing user stories, developing system acceptance tests, setting priorities and answering questions about the requirements. The clarity and understandability of requirements are improved because of the immediate access to customers and their involvement in the project when needed. However, from a requirements honesty viewpoint, agile development may have a negative impact on the requirement principles of purposefulness, appropriateness and truthfulness (Pinheiro, 2002). For example, XP relies

[1]The terms ‘requirements analysis’ and ‘requirements engineering’ are used synonymously in this paper as is commonly done in the RE literature.

almost completely on oral communication with the customer. Among the 14 assumptions underlining agile development (Turk *et al*., 2005), the following four are related to the RE process:

- The changing-requirements assumption: Requirements always evolve because of changes in technology, customer needs and business domains. Therefore, agile development does not spend much time in initial requirements elicitation. Instead, requirements emerge during the development process.
- The documentation assumption: Developing extensive documentation and models is counterproductive. Therefore, agile development does not document requirements in detail upfront.
- The customer interaction assumption: Customers are available for frequent interaction with the developers. Agile development relies on these interactions to elicit and validate requirements.
- The cost-of-change assumption: The cost of making changes to the system does not dramatically increase over time. As evolving requirements do not increase the development cost, RE is carried out iteratively and incrementally.

These assumptions are different from the assumptions underlying traditional development. The assumptions underlying RE in traditional development include (Sillitti *et al*., 2005) the following:

- The customer is able to specify all his/her needs in the initial phase.
- One or more stakeholders are in charge of the requirements gathering activity.
- The development team can readily understand customer needs.
- The structure of the organizations is hierarchical, and there is a sharp separation of the different functions.

A recent survey (Sillitti *et al*., 2005) on how agile and traditional approaches manage requirements uncertainty suggests that they differ in their approaches to managing uncertainty and requirements gathering. Agile development relies on the interaction with the customer and gathers requirements throughout the development process. In contrast, in traditional development, requirements gathering occurs primarily at the beginning of the project. Proponents of agile methods suggest that because of constant interaction with the customer throughout the development process, they are likely to achieve higher customer satisfaction. However, the informal nature of agile RE practices may be considered unacceptable (Nawrocki *et al*., 2002) when assessed against the Software Engineering Institute's (SEI) Capability Maturity Model and the Sommerville–Sawyer Model (SSM) of RE (Sommerville & Sawyer, 1997).

Several approaches to complementing agile RE with traditional RE methods have been proposed to address this problem (Boehm, 2000; Grünbacher & Hofer, 2002; Paetsch & Maurer, 2003). These include the use of an explicit requirements negotiation (Grünbacher & Hofer, 2002), establishing traceability (Lee, 2002), incorporating aspect-oriented concepts (Araujo & Ribeiro, 2005), or even incorporating an explicit RE phase (Nawrocki *et al*., 2002). The use of cooperative strategies for RE have been suggested for time-constrained environments (Jepsen, 2002).

Boehm (2000) provides four guiding principles that help tailor a requirements strategy to fit different situations: value-driven requirements, shared-vision-driven requirements, change-driven requirements and risk-driven requirements. These principles can be used to guide RE practices in dealing with developments of IKIWISI (I'll know it when I see it) and Commercial Off-The-Shelf (COTS) software as well as software in environments characterized by rapid changes in business and technology. Orr (2004) suggests that requirements should include descriptions of the outputs of the system and critical constraints. As the users may not be able to articulate these adequately, it is the job of the requirements analysts to 'invent' these requirements by working with them using techniques such as prototyping.

Much of the prior research is centered on assessing and improving agile requirements approaches as defined in popular agile methods such as XP and SCRUM. However, the presence of several confounding factors and the variations between prescribed and actual practices are sources of significant concern in such assessments of the effectiveness of agile methods (Erickson *et al*., 2005). Little is known about how RE is actually conducted in real projects in agile environments. Therefore, our research is motivated by the need to address the following questions about agile RE. What RE practices are adopted in software development in agile environments? What problems are encountered when these practices are used? Do these practices mitigate or increase requirements risks? The investigation of these questions can provide important insights into the current practices, help evaluate the practices suggested in the normative literature and suggest guidelines for RE practice in agile environments. Our research investigates these questions by examining RE activities in 16 actual agile projects.

RESEARCH FRAMEWORK

Prior studies have identified several factors that necessitate a new approach to achieve agility in software development (Cockburn, 2002; Levine *et al*., 2002). These factors include evolving requirements, rapidly changing technology and strict time constraints. In recent years, market forces, system requirements and implementation technologies have been changing at a steadily increasing rate. Many system development efforts are also geared towards creating innovative products, services and markets. The requirements for these systems may not be fully understood even by the customers (Beck *et al*., 1999; Levine *et al*., 2002). Domain experts may not exist for many of these 'new economy' applications or ways of doing business. Also, requirements are often vague at the outset and change even during development. As a result, it is impossible to follow the traditional RE practices and develop a specification that is clear, consistent and complete before the design and implementation begin. Throughout the development process, major changes to requirements may be needed. Rather than relying on formal specifications that are constantly changing, users and sponsors rely on informal dia-logue to convey their requirements to the developers. Furthermore, the ability to get products early to market is often considered critical for success in high velocity environments. Reducing the cycle time for software development is frequently considered the highest priority in agile

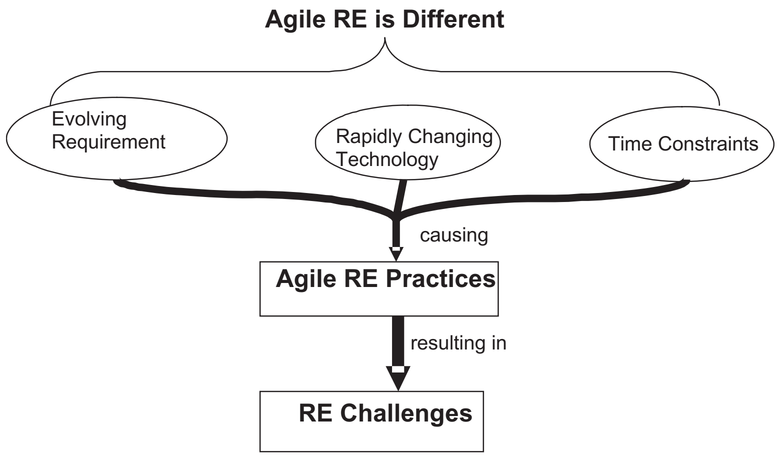

Figure 1. Research framework.

projects. A lengthy requirements analysis phase is considered to hinder the speed of development, and therefore, lightweight practices such as prototyping are adopted to quickly build an application.

The factors just mentioned have caused the adoption of a different set of practices in agile RE. These practices also raise some critical challenges to successful software development. For example, the reliance on oral communication and lack of documentation make it difficult to trace the evolution of the application over time. Our research seeks to identify both the practices characterizing agile RE and the challenges that result from these practices. The framework shown in Figure 1 guides our research approach described in the next section.

RESEARCH METHOD

Research design

Our research takes an interpretive multi-case study approach based on the following ontological and epistemological assumptions (Orlikowski & Baroudi, 1991; Mason, 1996). We believe that requirements analysis is a social–political process that depends largely on human actions and interactions and is influenced by many contextual factors. This phenomenon can only be understood through examining it in its settings. The case-study research strategy that focuses on understanding the dynamics present within a setting (Eisenhardt, 1989) and attempts to examine a phenomenon in its real-life context is appropriate for the study of requirements analysis in agile development. Our study seeks to develop a deep understanding of this process in terms of the unique set of practices followed and the challenges that result from them.

Data sources

We used a purposeful sampling strategy in this research (Miles & Huberman, 1994). Data were collected from 16 organizations that develop software using agile methods. These organizations are located in three major metropolitan areas in the USA. Our objective was to understand how and why agile requirement analysis differs from traditional requirement analysis. The study was conducted in two phases.

In the first phase, we conducted case studies of 10 organizations that characterize themselves as involved in agile or high-speed software development. Several practices that characterize the high-speed software development process emerged from this phase. While these organizations did not explicitly follow any specific ‘brand’ of agile methodology, the observed RE practices were quite similar to those suggested by agile methods such as XP and SCRUM. In the second phase, we collected data from six organizations that used XP and/or SCRUM.

The organizations that participated in the study represent a rich mix of industries, from health care to software consulting. Data were collected through multiple methods: semi-structured interviews, participant observations and review of documentation. Within each organization, we interviewed a variety of stakeholders, including top management, product managers, quality-assurance personnel, software developers, senior architects and project mangers. A summary of the characteristics of the study participants is presented in Appendix I.

Interviews were semi-structured and were focused on RE practices and issues faced in following these practices. The average duration of the interviews was about 2 h. For each interview, the flow of questions and the scope and depth of each topic were adjusted to fit the interviewee’s role in the development process. Additional informants were identified during the interviews and were included in the study to provide further insights. All interviews were tape-recorded and transcribed. Appendix II contains an excerpt from the interview guide used in the study. Interesting issues observed during the interviews were discussed during follow-on interviews. We also reviewed requirements documents such as story and task cards whenever feasible.

Analysis procedure

Data analysis was conducted in parallel to data collection (Miles & Huberman, 1994). Results from preliminary data analysis guided further data collection. Whereas most interviewees focused on the current or recent project experiences, their responses included information on multiple projects from previous experience. Therefore, each firm was the unit of data analysis. Three types of coding techniques commonly used in grounded theory research were used in the analysis of data: open coding, axial coding and selective coding (Strauss & Corbin, 1990).

The data analysis began with open coding. Even though a framework was used to guide the research, the open coding assumes that there was no prior assumption about what the categories should be. The incidents, events, quotes and other instances gathered during data generation were examined and compared for similarities and differences. Groups of data were identified and labelled as categories and subcategories.

After open coding, axial coding was conducted to achieve the theory-generation objective of this research. Data were analysed again to uncover relationships among categories and subcategories. This analysis resulted in the identification of concepts.

After identifying general relationships among concepts, we conducted selective coding, which refers to finding larger patterns by selecting a core category, systematically relating it to other categories and validating those relationships. Additional interviews were conducted to gain further clarifications on some concepts. Patterns were refined and were associated with agile RE processes. The analysis identified core categories and their interrelationships, explaining how and why agile RE is different from traditional approaches. To generate theoretical insights from this analysis, we focused on similarities and differences among the 16 organizations, identifying variables and relationships about agile RE practices.

Two coders separately coded the data and the results were compared. Observed differences were resolved after detailed discussions. The data was re-coded to arrive at a set of practices that are common across these organizations.

RESULTS

We identified six agile RE practices and seven challenges to RE from our study (Figure 2). In the next section, we present an account of the core story line and the relationships between categories and subcategories found in our analysis. We discuss the agile RE practices with examples drawn from our study. We then present details on the challenges that these practices pose for the RE process. Following this presentation of the results from the study, we present the agile practices and challenges across all the firms for comparison. Then, we discuss how these practices mitigate as well as exacerbate critical risks in RE activities.

Agile RE practices

We identified six agile practices: face-to-face communication over written specifications, iterative RE, managing requirements change through constant planning, extreme requirements prioritization, prototyping, and review meetings and tests. These practices are discussed in detail in this section.

1. Face-to-face communication over written specifications

Agile software processes prefer face-to-face communication over written specifications. The focus of RE in an agile environment is to effectively transfer ideas from the customer to the development team, rather than creating extensive requirements documents. A senior manager explains this practice as follows:

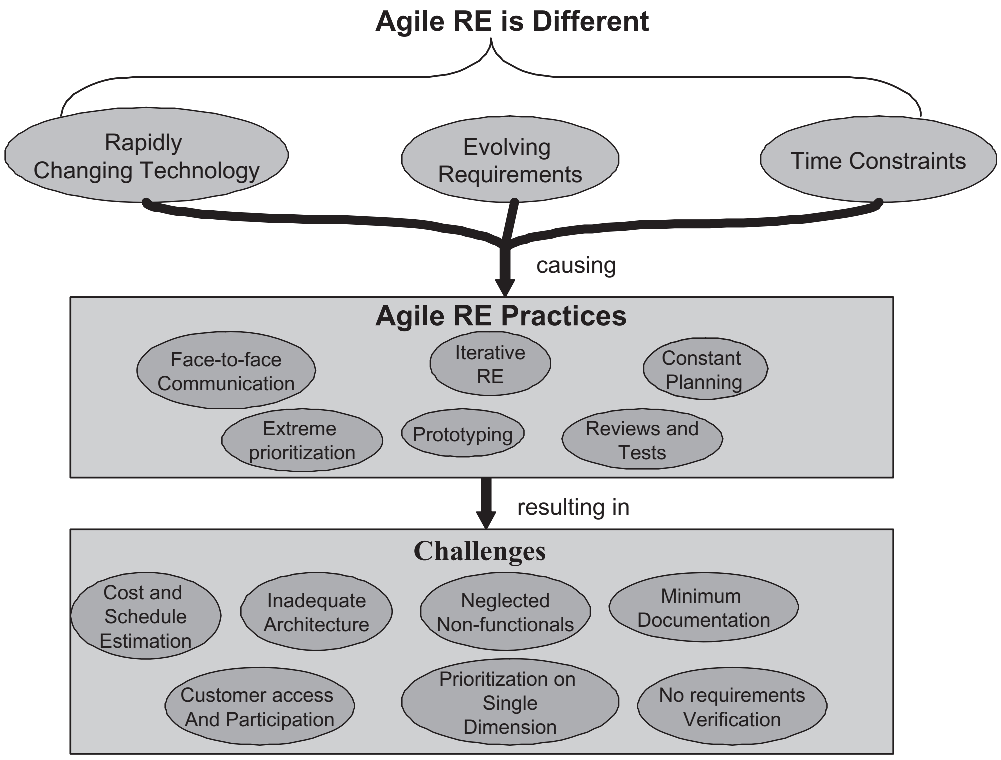

Figure 2. Agile RE practices and challenges.

> . . . we showed the customer what we built based on the requirements documents. We got feedback from customer. They said, 'this is OK, this is not what we want in the system'. These sessions were helpful to get the customer to say how they want things to work. (ServeIT)[2]

Simple techniques such as user stories are used to define high-level requirements. These short and abstract descriptions serve mainly as anchors for future discussion. Project managers use them to estimate project costs and schedules. Requirements are discussed in detail with the customers before and/or during the implementation. However, formal documentation of the specifications is seldom created. One exception is BankSoft, a banking software company in which requirements are pre-defined and formal documentation is mandatory. However, even for such mission-critical applications, face-to-face communication with the customer is considered important. For example, BankSoft held a daily meeting with the product manager, who served as a surrogate customer to discuss the requirements and alternative

[2]This is a direct quote from a subject participating in our study. Henceforth, the pseudonym for the subject's organization will be identified within parenthesis.

solutions. As commented by the system architect, 'I need a surrogate for a customer if nothing else . . . because think about periodically trying to get in touch with the business analyst . . . it really has to be everyday. It has to be all the time' (BankSoft). The formal documentation of requirements could not eliminate the need for frequent communication as 'everything is ambiguous, if you give me exactly what the customers want, they (the customers) are going to look at it and say, that's neat (but) I want something different' (BankSoft).

The effectiveness of this practice heavily depends on the intensive interaction between customers and developers. The customer needs to be collaborative, representative, available, capable and knowledgeable (Boehm, 2003) to provide the right requirements to the development team. For projects that cannot achieve such high-quality interaction, this approach poses high risks, including inadequately developed requirements or, worse still, wrong requirements.

Agile requirement analysis heavily relies on customer involvement to provide detailed requirements. Most agile methods require co-located customers (for example, XP mandates it as a core practice). In comparison with projects that have infrequent customer engagement, these practices lead to better project outcomes (Cao *et al*., 2004). However, it is very difficult to implement this practice (Lindvall *et al*., 2002). Most software projects we studied do not have direct access to a customer representative. If a real customer is not available, product managers or business analysts, who have direct contacts with customers, act as surrogates. These surrogates meet with the development team frequently to discuss changes in requirements and to make decisions on development options. A senior manager and a customer representative explained how a customer co-located with the development team provides opportunities for quick learning:

> . . . I (the product manager) actually moved my cube and office to the developers' area. . . . I was here with one of the cube with the guys. So we would have the discussion here and they would go back and work on the implementations. and likewise, I will work on the customer interactions, right there from the same location. We didn't do a lot of scheduled meetings like that stuff; we were all at the same building on the same project. (EBizco)

A short daily meeting between the team and the customer is another mechanism used to enhance communication. In some firms, all the team members attend the meeting, whereas in others only the project manager, technical lead or developers who have questions meet the customer. These meetings were considered very valuable by the participants.

2. Iterative RE

In agile development, requirements are not pre-defined; instead they emerge during the development process. Initially, high-level requirement analysis is carried out at the beginning of the project. At this time, the development team acquires a high-level understanding of the critical features of the application, rather than creating a detailed specification of requirements. Furthermore, these requirements are not intended to be complete or cover all the features of the system. Instead, they serve as a starting point to plan the initial release cycle of the project. As more is known about the product, more features or user stories are added. There are several reasons for commencing development without spending much time on requirement analysis initially, including: (1) the requirements volatility is high; (2) even if the business domain is stable, technical detail are unknown and will affect implementation, requiring re-evaluation of requirements; and (3) the customer can only clearly define the requirements when they see it ('I will know it when I see it' – IWKIWISI). One developer contrasts this approach with the waterfall approach in which

> . . . We basically spent about two months on upfront analysis and design, without even touching code. Once we got into the code, of course you learned what you came out at the end didn't match what you designed up front. There was a lot of waste of time up front. (SecurityInfo)

Agile requirements analysis continues at each development cycle. At the beginning of each development cycle, the customer sits down with the development team to provide detailed information on a set of features that need to be implemented. During this process, requirements are discussed at a greater level of detail. Also, very often, requirements analysis is intertwined with the design activity. As a result, the outcomes are a set of finely defined requirements, a preliminary design and sometimes an implementation plan.

Although iterative requirements analysis is often associated with dynamic business environments where requirements change frequently, our study shows that this approach may be appropriate even in stable business environments where the changes often come from unforeseen technical details, especially when adopting new technologies. One developer talked about her experience on changing requirements based on the evolution of the technology:

> I think another issue that caused the problem (requirements change) is because we are using the latest technology. We actually use c# and .net. When we first got started, we even used the beta version of the.net framework and visual studio. So at that time, they really didn't have any guidelines at all. Right now, Microsoft® gets so involved and they have the whole set of best practices, guidelines, architecture suggestions. (FinCo)

3. Requirement prioritization goes extreme

Prioritization is essential in requirements analysis, especially for projects that have limited resources in terms of budget, staff and schedule (Sommerville & Sawyer, 1997, Schneider & Winters, 1998). Requirements can be categorized from the standpoint of importance into essential requirements, useful capabilities and desirable capabilities (Sommerville & Sawyer, 1997). Setting priorities early in the project helps deliver the most valuable features as early as possible. However, few organizations have effective practices for assigning and modifying priorities or communicating them to project members (Lubars *et al*., 1993).

In agile software development, a common practice is to implement the features with the highest priority early in the project so that the customers can realize most business value. During development, the customer's, as well as developer's, understanding of the project improves and new requirements are added, or existing requirements are modified. To keep priorities up to date, prioritization is repeated frequently during the entire development process. One developer described this experience as follows:

> We were delivering high value every step of the way. And what you got at the end of the project, basically working on the medium value stuff and we leave the lower value stuff, and when we got the project end, they [the customer] were very happy about what they had, they were like 'we could ship this months in advance'. So as a result, you don't waste [effort] on things that aren't important. So we got more value out of it . . . We were able to leave out the unimportant stuff. (BankSoft)

We observed at least two important differences between the traditional and agile RE in the way requirements are prioritized: (1) continual re-prioritization of requirements; and (2) criteria used in prioritization.

In traditional RE, requirements are prioritized typically once at the requirement analysis phase. In contrast, prioritization of requirements is conducted in each development cycle during agile development. Moreover, requirements are prioritized together with other development tasks such as making changes to existing functionality, bug fixings and sometimes refactoring. Re-prioritization happens as part of the planning activity in each development cycle. However, in traditional development, re-prioritization is more difficult, as explained by a product manager.

> I would say in that particular project [a traditional project], we didn't really have [re-prioritization] . . . because of the politics, because the formality of the relationship with the engineers . . . often there are external influences, like a chief executive officer set 12/01/02 is your day, we need to ship the product by this particular date. So then it's back to the engineering director, the engineering manager, and he says 'we need to have requirements locked by this particular date' and if we were to alter the requirements after that date, it has a direct impact on the deliverable date . . . So there wasn't a lot latitude to re-prioritize or change the requirements on this project once we were underway. (SecurityInfo)

Another difference involves the factors that drive requirements prioritization. In traditional development, it may be driven by many factors such as business value, risks, cost, implementation dependencies, etc. (Firesmith, 2004) Both the customer and the developer provide inputs to this process. The customer identifies the features that provide them with the greatest benefit. Developers identify technical risks, costs or implementation difficulties. In contrast, agile RE tends to consider predominantly a single factor – business value as defined by the customer. As customers are highly involved in the development process, they can provide business reasons for each requirement at any development cycle. Such a clear understanding of the priorities of the customer helps the development team better meet customer needs.

4. Managing requirements change through constant planning

A core principle in agile software development is to adapt and react quickly to changes demanded by the environment. Accommodating changes in requirements during the development process is a way of tuning the system to better satisfy customer needs. Changes are easier to implement and cost less in agile development, says a developer.

> Planning is a constant activity . . . it's constantly being revisited as these things change. Because we don't make fixed plans, and try to conform to them, accommodating change is easier. (NetCo)

In agile development, generally two types of changes are commonly made: (1) features are added or dropped; and (2) changes are made to features already implemented. At the end of each cycle, tests are used to evaluate the delivered features. Customers provide feedbacks and request changes if their expectations are not met. In our study, interestingly enough, we found that this kind of change was rare. As one developer mentioned,

> It's often that you didn't quite get the story implemented 'right', and there is feedback throughout the iteration which ensures that you don't get off track. So it's usually more a case of tweak . . . spelling, little graphical things . . . , e.g. color, positioning . . . (AgileConsult)

When asked about the magnitude of changes on delivered features (stories), he said, '. . . it's usually tiny things . . . it's typically light weight stuff. A significant change . . . maybe 2 in 10' (AgileConsult).

Such a low occurrence of post-development change is interesting because this ability is often touted as an important benefit of agile processes. Our study finds that the reason for this is the frequent communication between the developer and the customer during development. Before implementing a feature, the developer engages in detailed discussions with the customer to roughly understand what the customer needs. Also, they get constant feedback from the customer as the features are implemented. With this intense interaction, the need for major changes to the delivered features is likely to be very low. In the absence of this interaction, major changes may be necessary, as experienced by one developer.

> The product management department gives us feedback only after release. But once you release to end users, it becomes much harder to do big rewrites like that so everything starts going badly. (AgileConsult)

5. Prototyping

One of the fastest ways to settle requirements specifications seems to be the development of a prioritized list of features. Instead of using formal requirements documents, many projects use prototyping as a way to communicate with their customers. Based on customer feedback on the prototypes developed for demonstration and experimentation, requirements are validated and refined.

> We are supposed to have full requirements and design documents but a lot of programmers use the prototype and go back and forth to check, or go back and ask: what was this supposed to do. (HuCap)

> We need to have a vision to know what will be the next steps because the first solution is often a prototype. Prototyping is important. (Venture)

> This helps reduce the margin of error. Piloting applications and releasing them to end users in iterative fashion are other useful practices. (ServeIT)

To a certain extent, the production software itself can be a form of an operational prototype, a refinement of the code created for experimentation with required features. The rush to market encourages a tendency to deploy these prototypes rather than wait for robust implementations of the desired features. The ability to quickly deploy newer versions of the product (enabled by the web architecture for delivery) is also another factor promoting this practice. However, in some organizations there is recognition of the risks involved in deploying prototypes in production mode. Besides causing problems with issues such as scalability, security and robustness, prototypes may not be easy to maintain or evolve.

> We often build the prototype and start including functionality on top of prototype. It works because of some sharp people. We are working on a blue print for a more structured process. (Transport)

> Our prototypes worked fine at the beginning, but they didn't scale well. It was also difficult to maintain them or support them beyond a few iterations. (Entertain)

6. Use review meetings and acceptance tests

Agile approaches use frequent review meetings for requirements validation. At the end of each development cycle, a meeting that involves developers, customers, Quality Assurance (QA), management and other stakeholders is scheduled. During the meeting, the features delivered are demonstrated; customers and QA ask questions and provide feedback and comments. The review meetings show that the project is on target and within schedule, increase customer trust and confidence in the team, and highlight problems early. In SecurityInfo, a developer commented on the use of review meetings:

> . . . [the product manager who is a surrogate customer] likes to be able to being involved . . . give good feedback each iteration, because we basically would have a demo, we bring PM [product manager], QA, everybody was involved. The engineers would just show them what we did. And we got good comments from everybody. QA wanted to bring up something how we designed it, we could take that back. (SecurityInfo)

However, in many organizations that participated in the study, only minor issues are discovered during review meetings. Requirements validation is largely carried out at the iteration planning meeting, where the team communicates with the customers about detailed requirements. As described by the project manager at SecurityInfo,

> We basically get some minor feedback. QA might say 'that's a neat feature, if you add this to it, it might add some more value on it' . . . The big thing was when, partly the iteration

planning, remember I said, at the start of the iteration, I will sit down with the PM for design and we would talk about features. PM sometimes brought up new things he found out as he talking to more customers, so he might bring new feature to the table . . .' (SecurityInfo)

Even though the purpose of the review meetings is to 'review' the developed features and obtain feedbacks from the customer, they are primarily designed to provide progress reports to the customer as well as other stakeholders in the organization. Our study suggests that requirements validation is largely carried out by frequent communication with the customer.

Acceptance tests developed by the customer, sometimes with help from QA, are another means used in validation and verification. These tests are treated as a part of requirements specifications in some organizations.

Another technique commonly used to validate requirements is Test-Driven Development (TDD), an evolutionary approach in which tests are developed before a new functional code is written (Astels, 2003; Beck, 2003). TDD treats writing tests as a design activity in which the behaviour of code is specified by a test.

Having those tests allow you to be more adventurous in terms of making changes and trying out ideas . . . you get very quick feedback if it goes wrong . . . you write code that talks about what the system's behavior should be. (AgileConsult)

In summary, our study suggests that agile RE practices do not follow traditional RE principles or guidelines defined in the RE literature (Sommerville & Sawyer, 1997; IEEE, 1998). An examination of these principles reveals that the fundamental assumptions behind traditional RE principles are not valid in the agile environment. For example, the IEEE Recommended Practice for Software Requirements Specifications (IEEE, 1998) states that

This recommended practice . . . is based on a model in which the result of the software requirements specification process is an **unambiguous** and **complete** specification document. It should help (1) software customers to **accurately** describe what they wish to obtain; and (2) software suppliers to understand **exactly** what the customer wants . . . (bold emphasis in text originally)

In contrast, the agile development is often done in an environment where it is impossible or even inappropriate to develop unambiguous and complete requirement specifications. These fundamental differences have led to a set of agile RE practices. Instead of following a formal procedure to produce a complete specification that accurately describes the system, agile RE is more dynamic and adaptive. Agile RE processes are not centralized in one phase before the development starts; they are evenly spread across the entire development process.

Agile RE challenges

Whereas the benefits of agile RE practices have received much attention, our study reveals that these practices pose several challenges that must be carefully addressed by development organizations to achieve success. In this section, we discuss some of the critical challenges identified in our study.

1. Problems with cost and schedule estimation

The agile approach towards RE makes the estimation of costs and schedules more difficult than with traditional methods. The importance of accurate estimates is highlighted by the observation that different estimates in effect guide the development team to produce different products (Abdel-Hamid & Madnick, 1991). However, it is difficult to develop accurate estimates of costs and schedules during the early stages of a project in agile software development that is characterized by unstable problem domain, requirement volatility, and dynamic planning and design phases. Furthermore, these estimates are adjusted over time during the development process.

Projection of completion dates is discouraged in the agile literature because embracing change renders these dates useless (Beck *et al*., 1999). Because agile approaches do not have a formal requirement analysis phase, the initial estimation of project size is based on the known user stories. This estimate serves as a starting point and will be updated frequently. The planning process is carried out at a coarse level. Known functionalities are broken into pieces and each piece is estimated. Many of these may be discarded and many more may get added during development. One agile developer described the process as follows:

> So during the early iterations I add 50% to any estimates whenever anyone is computing a project completion date. Once we have a few iterations under our belt and our velocity is stable, I drop the multiplier down to 25%. (SecurityInfo)

However, it should be noted that the short cycle time and frequent feedback help an agile development team create better estimates for each iteration, especially after gaining experience over several cycles. This suggests that agile development helps create better cost and schedule estimates for individual development cycles, but because the scope of the project is subject to constant change, it is difficult to create accurate cost and schedule estimates for the entire project.

2. Inadequate or inappropriate architecture

The architecture chosen by the development team during the early cycles may become inappropriate or inadequate as newer requirements become known. Rework of the architecture may add significantly to project cost. FinCo reported this problem.

> . . . We wanted to get this done as soon as possible so the architecture was not very scalable, sometimes it was very hard to extend. It was a concern for us and that's why for the second time we rewrote some of the code. (FinCo)

AgileConsult had a similar experience:

> My current project has a code base that was not well designed, now it is like a great big hairball: you try to pull anything and all you get is knots, so I have to rewrite most part of it. I estimate at 2–3 months or more effort. (AgileConsult)

Refactoring is a practice that is used to change the internal structure of the software to make it easier to understand and cheaper to modify without changing its observable behaviour (Fowler, 1999). Some agile methods such as XP rely on refactoring to improve the overall architecture of the system. However, our study suggests that the need for refactoring is not always obvious and the ability to do refactoring depends on many factors such as the experience of the developers and schedule pressure. Moreover, our study observes that refactoring, as an ongoing activity to improve the design, often could not completely address the problem of inadequate or inappropriate architecture. As explained by the developer in AgileConsult,

> . . . the most important factor is probably whether the code you are trying to refactor has become legacy or not. If it is legacy code, then forget refactoring . . . it is impossible. I am actually experiencing that now. My current project has a code base that was not well designed . . . the rewrite is the result from lack of OO design skills and inappropriate architecture. (AgileConsult)

As the above experience suggests, relying on refactoring to address problems with the architecture may become too expensive. Occasionally, the only alternative is to throw away the code and rewrite the entire system or modules. The developer in AgileConsult quoted above mentioned his experience with rewriting large application modules (200–330 000 lines of code) about five times. When asked whether agile practices provide any solution to this problem, the developer answered as follows:

> I'm sorry but I do not think there is an effective solution to this problem. Yes it helps to continuously refactor, but then you have to always have a team that will be able to continuously refactor, and you have to have a management team that is courageous enough to allow the developers to continuously refactor and so on and so forth . . . in my experience, management will always be scared to change production code . . . a manager will say it is good enough, but once it is good enough and refactoring stops then it becomes legacy and 'good enough' may become 'not enough' later on and you have to rewrite later, but later, means expensive. (AgileConsult)

3. Neglect of non-functional requirements

A major concern with iterative RE in agile development is the inadequate attention given to non-functional requirements. They are often ill defined and ignored during early development cycles. Customers often focus on core functionality and ignore issues related to scalability, maintainability, portability, safety or performance. A product manager lists various quality factors that were ignored in their development: 'many factors – usability, robustness, scalability, portability, maintainability and all these ilities' (Entertain). 'Overall stability is also important – We can't be crashing. The system needs to be available and responsive. But, we don't test that . . . We have no specific test of stability. We just test for functionality and see if it stays up' (TravelAssist).

One common exception is the consideration of ease of use, especially when the customers are intensely involved in providing constant feedback on the evolving system. However, it should be noted that the development team has limited access to the customer. Therefore, even the evaluation of ease of use may not be very comprehensive. Many organizations suggested that the tendency to ignore critical issues such as security and performance early in the process has resulted in major issues as the system grows.

4. Customer access and participation

The effectiveness of communication between the customer and team depends on several factors, including: (1) customer availability; (2) customer consensus; and (3) customer trust, especially at the beginning of the project.

Indirect links between the customers and the developers through intermediaries or surrogates are less effective than direct links (Keil & Carmel, 1995). However, our study finds that onsite customer representation is difficult to attain. Of all the projects studied in Phase II, none of them had real onsite customers, but only used product managers as surrogates. In fact, only two projects had a full-time, onsite product manger while the others only had part-time access. One developer explained this difficulty:

> The best customers are always going to be busy with their jobs. Either they or their boss will not allow them to join your team full-time. Actually, in my experience even part-time is a problem. We were lucky to have about 3–5 h of [the product manager] per week. (HealthInfo)

When a system involves more than one customer group and each is concerned about different aspects of the system, achieving consensus or compromise within the short development cycle may be challenging. Also, each customer may not have a fully developed understanding of his/her own requirements because of the iterative nature of the process. Such fragmented view of the system that each customer may have is likely to negatively impact the project. One developer explains how such customer behaviour may hurt a project:

> Yes it did [hurt the project] . . . mostly in terms of each having their point of view and not wanting to compromise to speak with a single voice. On second thought there were some story writing sessions that were quite hostile . . . They'd argue about priorities of their set of stories as well. (NetCo)

The development team needs to spend extra effort to integrate the requirements of different segments through negotiation with each customer. For example, in the NetCo project, the project manager forced the customers to physically sit together to discuss requirements in order to achieve consensus.

Our study also finds that sometimes customers find it difficult to understand or trust the agile RE process. Prior research suggests that in software development, distrust hurts performance, whereas trust improves performance (Sabherwal, 1999). Distrust can lead to finger-pointing, as each organization or individual focuses on their own interests, seeking to identify how the others may hurt the project. In contrast, mutual trust leads participants to work together rather than seek ways to deflect blame. In agile development, trust between the customer and the developer is essential. It has been considered an ‘absolute prerequisite’ for any form of customer collaboration (Sharp & Robinson, 2003). We find that the establishment of trust between the customer and the developer can be very challenging in agile development. The customers may come from the traditional development background and do not understand or trust the development process that does not produce detailed requirements or design specifications. Our study finds that the customers’ attitude towards the process has a major impact on the project. One project at NetCo included three customer representatives, but only one of them had a positive opinion of the agile RE process. In this project, the project manager reported that the two customers who did not have high confidence in agile methods were not ‘good’ customers in terms of their ability to provide relevant information and feedback.

> Product manager was hostile to XP. They could never provide requirements fast enough. Actually it was not just fast enough, it was also that they didn’t give us very good directions and then expected us to be able to change everything very quickly . . . As a result, we rewrote the application in some way or another 3–4 times. (NetCo)

5. Prioritization on a single dimension

The business value-oriented approach to requirements analysis results in better understanding of the business domain and the system, prioritization of requirements, changes to design and elimination of unneeded functionality. These, in turn, help align the system better with business needs. However, using business value (often focused on time-to-market) as the only or primary criterion for requirements prioritization may cause problems such as an architecture that is not scalable or a system that is unable to accommodate requirements that may appear secondary at the beginning of the project, but become critical for operational success (such as security and efficiency requirements). The rush to coding has resulted in problems, claims an architect:

> We introduce a neat new widget that tickles back end system and there’s an instantaneous end rush. There’s no support to evolve. There is no opportunity to introduce a product slowly. Scaling has to be built in. (Transport)

6. Inadequate requirements verification

Agile RE focuses more on requirements validation than traditional approaches. However, it does not address aspects of verification as there is no formal modelling of detailed requirements. Consistency checking or formal inspections are seldom performed during agile RE. It has been argued that agile RE should include more detailed requirements verification into the process (Paetsch & Maurer, 2003).

The lack of clear or complete specification of several critical requirements sometimes makes development difficult. For example, a project at FinCo had to rewrite a major part of the system to implement a new feature. A developer told us, ‘We could have avoided that by thinking carefully . . . but because of the deadline, we just did it too quickly . . . we had to rewrite some pieces because it doesn’t work well’.

It should be noted, however, that agile practices place heavy emphasis on continuous testing. In fact, such testing is a core practice in agile methods such as XP. Whereas agile practices place much emphasis on acceptance testing, it is difficult to implement because of the difficulty of access to customers who can help develop these tests. Often QA plays this role.

7. Minimal documentation

Agile RE focuses on communication rather than documentation. Driven by the need for speed, development teams focus on the implementation of the functionalities rather than documenting requirements or design specifications.

> Following our methodology in the project is also important to ensure quality. However, especially in a high speed environment, we may not have the time to document a few things. (ServeIT)

While access to the right customer makes the need for documentation of specifications less important, close collaboration and proximity to other members of the development team often obviate the need for elaborate design documentation. However, when there is a breakdown in communication caused by a variety of problems, including the rapid turnover of personnel, rapid changes to requirements, non-availability of appropriate customer representatives and growing complexity of the application, the lack of such documentation may cause a variety of problems. These include the ‘inability to scale the software, evolve the application over time and inducting new members into the development team’ (ServeIT).

In summary, the seven challenges identified in our empirical study highlight the need for careful planning and evaluation necessary to make agile RE successful.

DISCUSSION AND ANALYSIS

To develop an understanding of the practices and challenges that were observed in the study organizations, we provide two different forms of analysis. First, we examine if and how the formal RE process activities presented in the RE literature are conducted in agile software development. Second, we develop a framework to compare both the practices and the challenges with RE risks described in prior research. With these analyses, we are able to distil two intractable problems in agile RE.

RE process activities in agile software development

Kotonya & Sommerville (1998) describe four sequential activities in the RE process: requirements elicitation, requirements analysis and negotiation, requirements documentation, and requirements validation. Our study shows that agile RE still includes the four activities, but these are implemented by different practices. First, agile RE activities are not sequential but are iterative and are performed during each of the several short development cycles. Second, there is no clear boundary between the four RE activities in the agile RE process. Instead, agile RE mingles all the four steps together. For example, during iteration planning, the requirements for the next iteration are elicited through communications with the customer and are 'documented' in story cards; also, the requirements are analysed and prioritized. These requirements are also validated by the customer during the meeting. Third, in agile RE, requirements are documented informally and no formal requirement documentation is created. As a result, requirements validation involves ensuring whether the requirements reflect the needs of the customer rather than focusing on the correctness of the requirements documentation.

In agile software development, requirement elicitation is carried out through iterative RE and face-to-face communication with the customers, whereas traditional requirements elicitation aims at discovering all the requirements before the development starts. It recognizes that requirements evolve over time and that requirements are discovered throughout the development process. Requirements analysis and negotiation are facilitated by iterative RE, face-to-face communication, constant planning and extreme prioritization. These agile RE practices help refine, change and prioritize requirements. Requirements are usually not formally documented in a specification document. Instead, they are documented informally as lists of features or stories. The detailed specification of requirements is replaced by the intensive communication between the development team and the customer. Finally, requirement validation is performed through review meetings and face-to-face communication. As discussed earlier, our study finds that while review meetings provide a formal channel to validate requirements, validation is often carried out through informal communication between the customer and the developers. In agile RE, requirements validation is focused on ascertaining whether the requirements reflect current user needs. The consistency and completeness of requirements are not formally checked as formal requirements documents are seldom developed. Our analysis suggests that the core of agile RE is informal and frequent communication. Only through communication are requirements discovered, analysed, documented and validated. Table 1 contrasts the traditional and agile approaches for performing the four RE activities. It also identifies the agile RE practices used to support each activity.

Comparison of agile RE practices and challenges

As a first step in developing a detailed understanding of the agile RE practices, we evaluated the degree to which each practice and challenge is adopted and followed in the 16 organizations studied in our research. A similar evaluation of the challenges that arise because of agile RE practices was also carried out. Each practice or challenge was scored as 'high', 'medium', 'low' and 'none' for each firm. Two coders analysed the data collected from the 16 organizations and assigned a value to each practice or challenge for each firm. A high interrater reliability was achieved. Also, discrepancies among the raters were discussed and resolved to arrive at a consensus. Our analysis shows that all practices are widely adopted among the firms in the study as most of the organizations are rated medium or high on all the practices (except for BankSoft). However, unlike agile RE practices, not all challenges are observed in

Table 1. Traditional and agile approach for requirements engineering (RE) activities

RE activities	Traditional RE approach	Agile RE approach	Agile practices used to support the RE activities
Requirements elicitation	Discovering all the requirements upfront	Iterative: requirements evolve over time and are discovered throughout the development process.	Iterative RE Face-to face communication
Requirements analysis and negotiation	Focus on resolving conflicts	Focus on refining, changing and prioritizing requirements iteratively	Iterative RE Face-to-face communication Constant planning Extreme prioritization
Requirements documentation	Formal documentation contains detailed requirements	No formal documentation	Face-to-face communication
Requirements validation	The consistency and completeness of requirements document	Focus on ascertaining whether the requirements reflect current user needs	Review meetings Face-to-face communication

all the firms. The degree of the challenges is also more diversified. The customer availability and concurrence among customers are the most common challenges that are observed in almost all firms.

Risks in RE

Requirements engineering is associated with a variety of risks (Curtis *et al*., 1988; Barki *et al*., 1993; Walz *et al*., 1993; Kraut & Streeter, 1995; Keil *et al*., 1998). Davis (1982) lists three uncertainties with respect to RE: (1) uncertainty with respect to existence and stability of a set of requirements; (2) uncertainty with respect to users' ability to specify requirements; and (3) uncertainty with respect to ability of analysts to elicit requirements and evaluate their correctness and completeness. These uncertainties are affected by many factors, including the stabilization and complexity of the system, single or many users, user experience, as well as the analysts' experience.

Lawrence *et al*. (2001) identify the following critical risks involved in RE practices: (1) overlooking crucial requirements; (2) inadequate customer representation; (3) modelling only functional requirements; (4) not inspecting requirements; (5) attempting to perfect requirements before beginning construction; and (6) presenting requirements in the form of designs. DeMarco & Lister (2003) discuss five core risks in software development of which three relate to RE: (1) intrinsic schedule flaws; (2) specification breakdowns; and (3) scope creep.

Not surprisingly, these 12 delineated risks from the literature are slightly repetitive and overlapping. We have consolidated these into nine distinct risks (with minimal overlap) in Table 2. This table also aligns these risks with the agile RE practices and the agile RE challenges that were observed in our study and presented in the previous sections. The agile

Table 2. Mapping risks and agile requirements engineering (RE) practices and challenges

RE Risks	Mitigation by agile RE practices	Exacerbation by agile RE challenges
Lack of requirements existence and stability (Davis, 1982; Lawrence *et al*., 2001; DeMarco & Lister, 2003; Keil *et al*., 1998)	Face-to-face communication over written specifications Iterative RE Managing requirements change through constant planning	
Issues with users' ability and concurrence (Davis, 1982; Lawrence *et al*., 2001; DeMarco & Lister, 2003; Walz *et al*., 1993; Barki *et al*., 1993)		Customer access and participation
Inadequate user–developer interaction (Curtis *et al*., 1988; Keil *et al*., 1998; Kraut & Streeter, 1995; Lawrence *et al*., 2001)	Face-to-face communication over written specifications	Customer access and participation
Overlooking crucial requirements (Lawrence *et al*., 2001; Davis, 1982)	Requirements prioritization goes to extreme Use review meetings and acceptance tests	
Modelling only functional requirements (Lawrence *et al*., 2001)		Neglect of non-functional requirements Inadequate/Inappropriate architecture
Not inspecting requirements (Lawrence *et al*., 2001)		No requirements verification Inadequate/inappropriate architecture
Presenting requirements in the form of designs (Lawrence *et al*., 2001)		Minimal documentation
Attempting to perfect requirements before beginning construction (Lawrence *et al*., 2001)	Iterative RE	
Intrinsic schedule flaws (DeMarco & Lister, 2003)	Managing requirements change through constant planning	Problems with cost and schedule estimation

RE framework presented in Figure 3 suggests that while some RE practices mitigate some RE risks, some challenges to RE that result from these practices also exacerbate some RE risks.

Analysis of agile practices, challenges and RE risks

Each of the nine distinct risks is either mitigated or exacerbated or both (mixed) by the agile RE practices and challenges identified in our research. While each of the six identified RE practices can mitigate some of the risks, it is not surprising to see that some of the risks are worsened by the use of agile practices. While it might seem elegant for the practices and challenges to map nicely to the risks in a one-on-one alignment, the real world of software development is too messy to support such a crisp mapping. In fact, each risk, to a varying degree, is mitigated by each of the practices and/or exacerbated by the challenges. However, we restrict our analysis to the most critical elements of this mapping (Table 2) to focus on how agile RE practices mitigate or exacerbate RE risks.

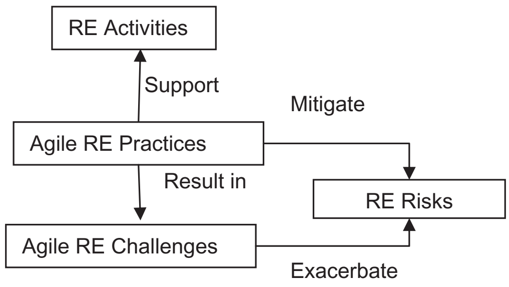

Figure 3. Agile RE framework.

In the discussion that follows, we provide the analysis of the relationship between risks, practices and challenges. This analysis permits us to characterize the risks in agile RE as either tractable or intractable. Tractable risks are those that are relatively easy to handle or manage in agile development. Intractable risks are those that are difficult to handle or manage when using agile RE.

The lack of requirements existence and stability

Agile RE practices have a positive impact on this distinct risk. Three practices (face-to-face communication over written specifications, iterative RE and managing requirements change through constant planning) help to stabilize and clarify requirements. Davis (1982) states that the uncertainty associated with requirements existence and stability increases in an unstable, poorly understood system that is undergoing change. This characterization is applicable to the agile development environment in general, and in the firms studied in this research in particular. Davis (1982) further proposes that iterative discovery methods are more appropriate in such situations to reduce the uncertainty of requirements existence and stability. In fact, this recommendation applies well to the agile RE practices (face-to-face communication, iterative RE and constant planning) followed by the firms in our study. These practices largely mitigate the risk, and consequently, we characterize this risk as tractable in agile RE.

Issues with users' ability and concurrence among users

When the user communities are fragmented, the limited understanding of the complete set of requirements possessed by the various groups makes it difficult to obtain shared understanding and more importantly concurrence among different customer segments on the importance of functionalities that are implemented during each development cycle. The short duration of the project cycle exacerbates this problem. Also, when the user representatives are not fully qualified – i.e. collaborative, representative, available, capable and knowledgeable – project teams face serious challenges in identifying appropriate requirements. Our study reveals that user's ability and concurrence among users are more critical in agile projects than in traditional projects as all RE activities (requirement elicitation, negotiation, documentation and validation) rely heavily on access to the right customer. Furthermore, our data indicate that this issue is of serious concern among a majority of firms in the study. Accordingly, we characterize this as an intractable risk in agile RE.

Inadequate user–developer interaction

Agile RE practices have both positive and negative impacts on this risk. Through the use of frequent customer interaction, participation of customers in the definition of requirements is improved. But this risk is also exacerbated if the customers do not trust the agile approach. Similarly, the narrow channelling of customer interaction through onsite representatives may actually limit customer interaction and thereby increase this risk. Also, as discussed earlier, the role of user–developer interaction in agile RE is considered critical, and its impact on the success of the project is enormous. So even though most of the firms adopted the face-to-face communication, our data also indicate that many firms face limited access to customers. Although the importance of user–developer interaction is well recognized by the participants in our study and supported by iterative RE and other related practices, the implementation of the user–developer interaction is very challenging. By making this interaction central to the development strategy, agile RE makes this risk tractable.

Overlooking crucial requirements

Agile RE practices have positive impacts on this distinct risk. The extreme focus on requirements priorities directly addresses this risk by focusing the attention of both the developers and the customers on the identification of crucial requirements. Review meetings, requirements prioritization and acceptance tests further draw attention to the most important requirements. These practices are adopted and followed by most of the firms in our study. Consequently, we characterize this risk as tractable in agile RE.

Modelling only functional requirements

Our study suggests that agile RE practices exacerbate this risk by often paying inadequate attention to non-functional requirements. With the exception of user interface design, agile RE tends to focus mainly on functional requirements. Of specific concern to a majority of the organizations in the study are operational scalability and security of systems. With the focus primarily on delivering functionality to the users as early as possible, these concerns typically do not receive much attention during early development cycles. Also, in the absence of a detailed design, the non-functional requirements were either typically not understood or explicitly specified at the appropriate time, or negotiated carefully to understand the trade-offs involved. The inadequate attention given to understanding and implementing non-functional requirements makes it harder to incorporate them as the system grows through successive development cycles. Also, without clear specification of the quality requirements, developers may make design choices that are arbitrary, and this makes it difficult to assess whether the system meets the real requirements.

While this risk is clearly exacerbated by the agile approach, it is interesting to note that it is often the customers, who are in control of the process, do not recognize the importance of non-functional requirements, especially early in the life cycle. In organizations such as Bank-Soft where the experienced developers anticipate and take into account key non-functional requirements such as security and scalability right from the outset, this risk is not serious. But in other organizations such as TravelAssist this issue has created serious bottlenecks and redevelopment of the system as non-functional requirements were not adequately addressed during the early cycles of the development process.

In summary, in majority of the organizations in the study, non-functional requirements were not understood or implemented adequately, and this risk is exacerbated to a serious degree. Consequently, we characterize this risk as intractable in agile RE.

Not inspecting requirements

Agile RE practices also worsen the risk of inadequately inspected requirements. The lack of detailed specifications and designs makes it difficult to do walkthroughs and inspections. This issue is especially important in situations where careful attention is not paid to non-functional requirements. Similar to problems with focusing only on functional requirements, despite the increased risk, our data show that the impact of this risk was mixed among the organizations studied. It was seen as a pressing concern in many organizations, but of very little concern in others. In the organizations that consider this a serious risk, inadequate design is also considered a big concern. For example, FinCo found that sometimes the design was not flexible enough to accommodate changes in requirements. In organizations where the new requirements and changes to existing requirements could be easily accommodated, this was not considered a major problem. In these organizations, changes to delivered features were rare because the requirements were examined carefully with the customer before they were implemented and the systems were relatively simple and well designed right from beginning. Given the mixed results, we do not have evidence to characterize this risk as an intractable problem in agile RE. Our study suggests that in relatively less complex systems with adequate attention paid to design, this risk is easily addressed. On the other hand, if the architecture is not appropriate, the risk is intractable.

Presenting requirements in the form of designs

Agile RE practices place minimal focus on formal documentation. Therefore, detailed requirements are often not formally documented, but are embedded implicitly in the code and designs. This practice, while efficient for speedy implementation, may impede the evolution of the software when changes to these requirements are necessary. Again, in this category, minimal documentation ought to increase risk, but our data show that for agile developers among the firms in the study, this was not generally believed to be a serious issue. Changes to requirements are expected and the development process is designed to ‘embrace change’ with practices such as iterative RE. Consequently, we characterize this risk as tractable in agile RE.

Attempting to perfect requirements before beginning construction

The agile RE practice of iterative RE is directly focused on the problem of perfecting the requirements. Multiple user reviews of requirements after each revision are implicit. There is a positive impact on this risk. In practice, our data show that this practice is adopted and followed by most of the firms in our study. Consequently, we characterize this risk as intractable in agile RE.

Intrinsic schedule flaws

Agile RE practices increase the risk of scheduling problems because of their inability to address cost and schedule estimation. The iterative nature of agile RE practices means that requirements emerge in an unstructured way. The requirements process is less predictable and thereby intrinsically difficult to schedule. On the other hand, the constant planning updates the schedule and cost estimation frequently based on the measured productivities. So schedule estimation is becoming more accurate over time. It should be recognized that iterative RE makes it possible for the organizations to be more accurate with estimates within development cycles. But the changing nature of the scope of the project makes it difficult to estimate costs and schedules for the entire project. Our data indicate that some firms had serious problems in estimating schedule and costs, while some did not see this as a big issue. Accordingly, we characterize this as tractable in agile RE.

Table 3 summarizes each of these risk analyses in terms of the relationship between RE risks, and tractability. Tractability depends upon whether the risk is related to an agile practice or challenge, and whether the case-study data indicate that the practice or challenge was important. For example, the first row shows how the risk that requirements would be unstable is mitigated by an agile practice involving constant planning. The data show that, among the firms in our study, this mitigating practice was prominent. This prominence of a mitigating factor leads to the conclusion that the problem is tractable. An opposite example is found in the risks related to users' ability and concurrence. This risk is exacerbated by issues with customers reported in our data, and the impact of this issue is shown to be high among the firms in our study. The serious concern among our firms over a risk that is exacerbated by agile RE leads to the conclusion that this risk is intractable.

CONCLUSIONS

Agile software development has recently received extensive attention among both practitioners and researchers. The primary focus with this approach is the delivery of a working system early in the development. Traditional requirements analysis practices have been characterized as too heavy for agile development, and a set of practices that foster an iterative and agile RE process has been proposed. Our study investigates how requirements analysis is conducted in 16 organizations that are involved in agile software development. We identify six distinct RE

Table 3. Characterizing tractability of risks in agile requirements engineering (RE)

RE risk	Agile practice or challenge	Impact of practice or issue	Degree of impact in agile practice	Character of problem
Lack of requirements existence and stability	Face-to-face Iterative RE Constant planning	Mitigates	Medium–High	Tractable
Issues with users' ability and concurrence	Iterative RE Customer access and participation	Mixed	High	Intractable
Inadequate user–developer interaction	Iterative RE Customer access and participation	Mixed	High	Tractable
Overlooking a crucial requirement	Requirement prioritization Review meetings and tests	Mitigates	Medium–High	Tractable
Modelling only functional requirements	Neglect of non-functional requirements	Exacerbates	Low	Intractable
Not inspecting requirements	No verification	Exacerbates	High/Low	Tractable
Presenting requirements in the form of designs	Minimum documentation	Exacerbates	Medium–Low	Tractable
Attempting to perfect requirements before beginning construction	Iterative RE	Mitigates	Medium	Tractable
Intrinsic schedule flaws	Cost & schedule estimate problems	Mixed	Low–Medium	Tractable

practices that are used in agile projects. These agile practices, while meeting the needs of agile software development, also pose seven distinct challenges to the RE process.

We have analysed these practices and challenges to understand their impact on risks in RE. Our study reveals that agile RE is different from the traditional RE as the former takes an iterative discovery approach. RE activities such as requirement elicitation, negotiation, documentation and validation are addressed together in each short development cycle. We also find that the intensive communication between the developers and customers is the most important RE practice in terms of its influence on each of the RE activities.

Our analysis enables us to conclude that agile RE practices are neither panacea nor poison to the risks intrinsic to RE. Not surprisingly, these practices mitigate some risks, exacerbate some risks and have a mixed impact on other risks. It might be too simple to say that agile RE practices have more positive impacts than negative through the sheer number of risks addressed. Only three risk areas are singularly mitigated by the practices, namely the lack of requirements existence, overlooking crucial requirements and the perfection of requirements before construction. There are three areas in which agile RE practices increase the known risks, namely overemphasis of functional requirements, inadequate inspection of requirements and incorporation of designs in requirements. Agile practices have mixed impact on the other three risks: user ability and concurrence, user–developer interaction and scheduling flaws. We characterize the risks in agile RE as either tractable or intractable based on whether the practices and challenges mitigate or exacerbate them. Tractable risks are those that are relatively easy to handle, while intractable risks are those that are difficult to handle or manage. Two intractable risks are identified: issues with customer ability and concurrence among customers, and modelling only functional requirements.

A primary contribution of our study is a framework that helps evaluate the appropriateness of specific agile RE practices. The framework describes the relationships between different agile RE practices and RE risks. This framework can be used by researchers to investigate in detail the components of each of the RE practices on the RE risks and hence project performance. This framework also suggests that each RE risk is impacted by a collection of agile RE practices. This framework provides a foundation for developing bundles or collections of practices to improve software development processes in specific contexts.

Our study suggests that agile RE practices provide a means for mitigating some risks. Therefore, a risk analysis of the development settings prior to requirements definition would help to determine whether agile RE practices or traditional RE practices would be more appropriate. For intractable risks, the traditional approach might be more appropriate. For tractable risks that are mitigated by agile practices, agile RE practices deserve serious consideration. Some risks are either exacerbated by agile practices or the impacts are mixed, but are still tractable. For example, if the development setting is known to have its most critical risks aligned with lack of requirements existence/stability and difficulty in detecting crucial requirements, then the setting may call for the use of agile RE practices. Alternatively, if the development setting is known to have its most critical risk aligned with poor customer ability/concurrence among customers, inadequate user–developer interaction, inadequately specified non-functional requirements, problems with requirements review, the tendency to frame requirements as designs, and scheduling problems, then the setting may call for the use of traditional RE practices. In settings such as these, agile RE practices may only worsen the project's risks. In summary, developers should carefully evaluate the risk factors in their project environment to understand whether the benefits of agile RE practices outweigh the costs imposed by the challenges.

REFERENCES

Abdel-Hamid, T. & Madnick, S.E. (1991) *Software Project Dynamics: An Integrated Approach.* Prentice Hall, Englewood Cliffs, NJ, USA.

Agile Alliance. (2001) URL http://www.agilealliance.com/ (last accessed 9 May 2006).

Araujo, J. & Ribeiro, J.C. (2005) Towards an aspect-oriented agile requirements approach. Eighth International Workshop on Principles of Software Evolution.

Astels, D. (2003) *Test Driven Development: A Practical Guide.* Prentice Hall PTR, Upper Saddle River, NJ, USA.

Barki, H., Rivard, S. & Talbot, J. (1993) Toward an assessment of software development risk. *Journal of Management Information Systems*, **10**, 203–225.

Beck, K. (2003) *Test Driven Development: By Example.* Addison-Wesley Professional, Boston, MA, USA.

Beck, K., Hannula, J., Hendrickson, C., Wells, D. & Mee, R. (1999) Embracing change with extreme programming. *IEEE Computer*, **32**, 70–77.

Boehm, B. (2000) Requirements that handle IKIWISI, COTS, and rapid change. *IEEE Computer*, **33**, 99–102.

Boehm, B. (2003) *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, Boston, MA, USA.

Cao, L., Mohan, K., Xu, P. & Ramesh, B. (2004) How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), Hawaii HI, USA.

Cockburn, A. (2002) Agile software development. Addison-Wesley Longman, Boston, MA, USA.

Curtis, B., Krasner, H. & Iscoe, N. (1988) A field study of the software design process for large systems. *Communications of the ACM*, **31**, 1268–1287.

Cusumano, M. & Yoffie, D. (2000) *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft.* Touchstone, New York, NY, USA.

Davis, G. (1982) Strategies for information requirements determination. *IBM Systems Journal*, **21**, 4–31.

DeMarco, T. & Lister, T. (2003) Risk management during requirements. *IEEE Software*, **20**, 99–101.

Eisenhardt, K.M. (1989) Building theories from case study research. *Academy of Management Review*, **14**, 532–550.

Erickson, J., Lyytinen, K. & Siau, K. (2005) Agile modeling, agile software development, and extreme programming: the state of research. *Journal of Database Management*, **16**, 88–99.

Firesmith, D. (2004) Prioritizing requirements. *Journal of Object Technology*, **3**, 35–47.

Fowler, M. (1999) *Refactoring: Improving the Design of Existing Programs.* Addison-Wesley, Boston, MA, USA.

Grünbacher, P. & Hofer, C. (2002) Complementing XP with requirements negotiation, XP. Third International Conference on eXtreme Programming and Agile Software Process in Software Engineering, Alghero, Sarding, Italy, 26–29 May.

IEEE (1998) IEEE recommended practice for software requirements specifications. *IEEE Standard 830*.

Jepsen, O. (2002) Time constrained requirement engineering – the cooperative way*,* International Workshop on Time-Constrained Requirements Engineering, Essen, Germany, 9 September.

Keil, M. & Carmel, E. (1995) Customer-developer links in software development. *Communications of the ACM*, **38**, 33–44.

Keil, M., Cule, P.E. & Lyytinen, K. (1998) A framework for identifying software project risks. *Communications of the ACM*, **41**, 76–83.

Kotonya, G. & Sommerville, I. (1998) *Requirements Engineering: Process and Techniques.* John Wiley and Sons, New York, NY, USA.

Kraut, R.E. & Streeter, L.A. (1995) Coordination in software development. *Communications of the ACM*, **38**, 69–81.

Lawrence, B., Wiegers, K.E. & Ebert, C. (2001) The top risks of requirements engineering. *IEEE Software*, **18**, 62–63.

Lee, M. (2002) Just-in-time requirements analysis – the engine that drives the planning game, XP. Third International Conference on eXtreme Programming and Agile Software Process in Software Engineering, Alghero, Sarding, Italy, 26–29 May.

Levine, L., Baskerville, R., Loveland Link, J.L., Pries-Heje, J., Ramesh, B. & Slaughter, S. (2002) *Software*. Engineering Institute, Pittsburgh, PA, USA.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F. *et al.* (2002) Empirical findings in agile methods, Second XP Universe and First Agile Universe Conference, Chicago, IL, USA, 4–7 August.

Lubars, M., Potts, C. & Richter, C. (1993) A review of the state of the practice in requirements modeling. IEEE International Symposium on Requirements Engineering.

Mason, J. (1996) *Qualitative Researching.* Sage, London, UK.

Merisalo-Rantanen, H., Tuunanen, T. & Rossi, M. (2005) Is extreme programming just old wine in new bottles: a comparison of two cases. *Journal of Database Management*, **16**, 41–61.

Miles, M.B. & Huberman, A.M. (1994) *Qualitative Data Analysis: An Expanded Sourcebook.* Sage Publications, Thousand Oaks, CA, USA.

Nawrocki, J., Jasinski, M., Walter, B. & Wojciechowski, A. (2002) Extreme programming modified: embrace requirements engineering practices. IEEE Joint International Conference on Requirements Engineering (RE′02), Essen, Germany.

Orlikowski, W. & Baroudi, J. (1991) Studying information technology in organizations: research approaches and assumptions, *Information Systems Research*, **2**, 1–28.

Orr, K. (2004) Agile requirements: opportunity or oxymoron? *IEEE Software*, **21**, 71–73.

Paetsch, F.E.A. & Maurer, F. (2003) Requirements engineering and agile software development*.* 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.

Pinheiro, F.A.C. (2002) Requirements honesty*.* 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Fachhochschule Mannheim, Germany.

Sabherwal, R. (1999) The role of trust in outsourced is development projects. *Communications of the ACM*, **42**, 80–86.

Schneider, G. & Winters, J.P. (1998) *Applying Use Cases: A Practical Guide.* Addison-Wesley, Boston, MA, USA.

Sharp, H. & Robinson, H. (2003) Customer collaboration: challenges and successes in practice: an agile development 2003 technical exchange. Agile Development Conference, Salt Lake, UT, USA.

Sillitti, A., Ceschi, M., Russo, B. & Succi, G. (2005) Managing uncertainty in requirements: a survey in documentation-driven and agile companies. 11th IEEE International Symposium on Software Metrics.

Sommerville, I. & Sawyer, P. (1997) *Requirements Engineering: A Good Practice Guide.* John Wiley & Sons, New York, NY, USA.

Strauss, A. & Corbin, J. (1990) *Basics of Qualitative Research. Grounded Theory Procedures and Techniques.* Sage Publications, Thousand Oaks, CA, USA.

Turk, D., France, R. & Rumpe, B. (2005) Assumptions underlying agile software development process. *Journal of Database Management*, **16**, 62–87.

Walz, D., Elam, J. & Curtis, B. (1993) Inside a software design team: knowledge acquisition, sharing and integration, *Communications of the ACM*, **36**, 63–76.

Biographies

Balasubramaniam Ramesh is a Professor of Computer Information Systems at Georgia State University. His research work has appeared in several leading conferences and journals, including the *IEEE Transactions on Software Engineering*, *MIS Quarterly*, *Journal of Management Information Systems*, *Journal of the AIS*, *Annals of Software Engineering*, *Communications of the ACM*, *Decision Support Systems*, *Information & Management*, *IEEE Computer*, *IEEE Software*, *IEEE Internet Computing*, *IEEE IT Professional* and *IEEE Intelligent System*. His research interests include supporting complex organizational processes such as requirements management and traceability with decision support systems, knowledge management, data mining and e-services. His work has been funded by several grants from leading government and private industry sources such as the NSF, DARPA, ONR, ARL and Accenture. He serves on the editorial boards of several leading journals and the programme committees of several conferences. He can be reached at bramesh@gsu.edu.

Lan Cao is an Assistant Professor of Information Technologies and Decision Sciences at Old Dominion University. Her major research interests are agile software development and software process simulation and modelling. Her work has appeared in several leading journals and conferences, including the *Communications of the ACM*, *Information Systems Journal*, *Journal of the AIS*, *IEEE Software* and *IEEE IT Professional*. She received a PhD degree in Computer Information Systems from Georgia State University. She can be reached at lcao@odu.edu.

Richard L. Baskerville is a Professor of Information Systems and Past Chairman in the Department of Computer Information Systems, Robinson College of Business, Georgia State University. His research specializes in security of information systems, methods of information systems design and development, and the interaction of information systems and organizations. His interest in methods extends to qualitative research methods. Baskerville is the author of *Designing Information Systems Security* (John Wiley & Sons) and more than 100 articles in scholarly journals, professional magazines and edited books. He is an Editor for the *European Journal of Information Systems* and serves on the editorial boards of the *Information Systems Journal, Journal of Information Systems Security* and the *International Journal of E-Collaboration.* Baskerville holds degrees from the University of Maryland (BS, *summa cum laude*, Management) and the London School of Economics, University of London (MSc, Analysis, Design and Management of Information Systems, PhD, Systems Analysis). He can be reached at baskerville@gsu.edu.

APPENDIX I CHARACTERISTICS OF STUDY PARTICIPANTS

Organization pseudonym	Industry and products	Number of employees interviewed	Organizational roles represented
EnCo	Energy and communications. Offers forecasting tools.	3	VP operations, project manager, software developer
HealthCo	Health care and utilities. Offers low prices for groups of customers.	6	President and CEO, VP technology operations, director of marketing research, CIO, developers

APPENDIX I cont.

Organization pseudonym	Industry and products	Number of employees interviewed	Organizational roles represented
Venture	Across industries. Offers to help brick and mortar companies develop web presence.	4	Director, chief financial officer, chief operations officer, developer
Entertain	Film and television industry. Offers high-tech indexing and searching tools online.	4	Project manager, marketing specialist, senior web developer, QA specialist
HuCap	Administration. Offers to carry out human resource administration for other companies online.	7	Project manager, architect, user interface design, web designers, web developers
TravelAssist	Transport and tourist industry. Offers services for these industries online.	6	Senior manager, project manager, QA manager, lead developers, web developers
ManageRisk	Across several industries. Offers insurance online.	3	Human resources manager, internet site manager, internet site developer
Transport	Transportation and logistics industry. Offers services for these industries online.	6	CIO, senior manager, project manager, architect, senior developer, web developer
ServeIT	Consulting and services. We studied the part of the firm that offers consulting services for business-to-business communication.	6	Senior manager, project manager, QA manager, QA specialist, web developers
HealthInfo	Healthcare information systems. Offers information systems solutions to hospitals, hospitals, physician offices and home health.	2	Senior software engineers
SecurityInfo	Security software. Offers software for internet security.	5	Software engineer, project lead, product manager, QA specialist
AgileConsult	Software consulting. Offers consulting services on software development. The company adopts agile methods.	2	Senior developer, project lead
EbizCo	Packaged software development. Offers e-business connections and transactions.	1	Senior software developer
FinCo	Online financial transactions support. Offers online payments.	1	Software developer
NetCo	Network software consulting. Offers services on developing network systems, and architectures.	2	General manager, senior software architect
BankSoft	Banking information systems. Offers software that handles financial transactions.	1	Senior software architect

APPENDIX II: EXCERPT FROM THE INTERVIEW GUIDE

General:
- Interviewee background information.
- Project Information: Domain (industry), functions, complexity and size, requirements volatility.
- Team Information: Team size, experienced vs. novice team members, turnover rate.

Agile process:
- Is any agile method explicitly used (e.g. XP, SCRUM)?
- Specific practices used.
- What are the practices that are different from traditional 'plan-driven' methodology such as waterfall?
- How are requirements identified? In what form? From whom?

Requirements elicitation and validation:
- Who is the customer? Do you have onsite customers?
- How do you evaluate the involvement and the competence of the customer?
- Factors influencing customer trust.
- How do you document requirements?
- Practices for validation and verification.

Requirements modelling:
- What techniques do you use to model the requirements?

Requirements volatility:
- How often and how much do requirements change?
- Proportion of tasks that change during development.
- Where and when did the changes come from?
- Factors affecting project scope.

Nature of requirements:
- Functional vs. non-functional.
- Criticality of requirements.
- Success factors for project.