Agile Requirements Engineering Practices: An Empirical Study

Lan Cao, *Old Dominion University*

Balasubramaniam Ramesh, *Georgia State University*

An analysis of data from 16 software development organizations reveals seven agile RE practices, along with their benefits and challenges.

The rapidly changing business environment in which most organizations operate is challenging traditional requirements-engineering (RE) approaches. Software development organizations often must deal with requirements that tend to evolve quickly and become obsolete even before project completion.[1] Rapid changes in competitive threats, stakeholder preferences, development technology, and time-to-market pressures make prespecified requirements inappropriate.[1]

Agile methods that seek to address the challenges in such dynamic contexts have gained much interest among practitioners and researchers. Many agile methods advocate the development of code without waiting for formal requirements analysis and design phases. (In this article, "requirements engineering" means the same thing as "requirements analysis," as is common in the RE literature.) Based on constant feedback from the various stakeholders, requirements emerge throughout the development process. Evolving requirements in a time-constrained development process cause the RE process for agile software development to differ from that for traditional development.

Few studies report on RE in agile development (see the related sidebar). Proponents present agile methods as a panacea for all the ills of software development, often focusing on the proposed practices' possible benefits.[2] Critics, on the other hand, have focused on the challenges that agile practices might present. In contrast, we've been systematically studying the agile practices that developers actually follow. Using a qualitative study of 16 organizations, we sought to answer two questions: What RE practices do agile developers follow? What benefits and challenges do these practices present?

How we conducted the study

Carolyn Seaman argues that software engineering's blend of technical and human-behavioral aspects lends itself to qualitative study.[3] Qualitative methods let you delve into a problem's complexity and develop rich, informative conclusions. For a relatively "uncharted land"[4] such as agile RE, a multisite qualitative case study approach is appropriate.

To understand how and why agile RE differs from traditional RE, we collected data from 16 organizations that employ agile approaches. (The "Study Participant Characteristics" sidebar provides details on the organizations. To protect their identities, we use pseudonyms.) These organizations are in three major US metropolitan areas.

The study had two phases. In the first phase, we conducted cases studies in 10 organizations that characterize themselves as involved in agile or high-speed software development. Although these

Related Work on Requirements Engineering in Agile Development

Although critics argue that agile software development approaches simply repackage established techniques,[1] others have recognized that requirements engineering in an agile environment is different.[2,3] For example, agile methods such as Extreme Programming (XP) advocate RE throughout the development life cycle in small, informal stages.[4] However, from a requirements honesty viewpoint, agile development might negatively affect the requirements principles of purposefulness, appropriateness, and truthfulness.[5]

In spite of the centrality of effective RE in agile development, the agile-RE literature is limited to a few high-visibility case studies (such as at Microsoft and Netscape)[6] and experience reports.[7] Much research has focused on assessing and improving agile requirements approaches as defined in popular agile methods such as XP and Scrum. Little is known about how real agile projects conduct RE. Also, it isn't clear whether developers are actually using the practices that agile methods prescribe.[7]

Recent studies have identified several problems that could result from the lack of detailed requirements specifications[8] and suggest several approaches to address these problems.[2,9,10] These approaches include using explicit requirements negotiation,[10] establishing traceability,[11] incorporating aspect-oriented concepts,[12] incorporating an explicit RE phase,[8] and using cooperative strategies for RE.[13] Some of these approaches might help mitigate the challenges that our study identified (see the main article).

References

1. H. Merisalo-Rantanen, T. Tuunanen, and M. Rossi, "Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases," *J. Database Management*, vol. 16, no. 4, 2005, pp. 41–61.
2. F.E. Paetsch, A. Eberlein, and F. Maurer, "Requirements Engineering and Agile Software Development," *Proc. 12th IEEE Int'l Workshops Enabling Technologies: Infrastructure for Collaborative Enterprises* (Wetice 03), IEEE CS Press, 2003, p. 308.
3. A. Sillitti et al., "Managing Uncertainty in Requirements: A Survey in Documentation-Driven and Agile Companies," *Proc. 11th IEEE Int'l Symp. Software Metrics* (Metrics 05), IEEE Press, 2005, p. 17.
4. K. Beck et al., "Embracing Change with Extreme Programming," *Computer*, vol. 32, no. 10, 1999, pp. 70–77.
5. F.A.C. Pinheiro, "Requirements Honesty," *Proc. 2002 Int'l Workshop Time-Constrained Requirements Eng.* (TCRE 02), 2002; www-di.inf.puc-rio.br/~julio/tcre-site/p3.pdf.
6. M. Cusumano and D. Yoffie, *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft*, Touchstone, 2000.
7. J. Erickson, K. Lyytinen, and K. Siau, "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research," *J. Database Management*, vol. 16, no. 4, 2005, pp. 88–99.
8. J. Nawrocki et al., "Extreme Programming Modified: Embrace Requirements Engineering Practices," *Proc. IEEE Joint Int'l Conf. Requirements Eng.* (RE 02), IEEE CS Press, 2002, pp. 303–310.
9. B. Boehm, "Requirements That Handle Ikiwisi, COTS, and Rapid Change," *Computer*, vol. 33, no. 7, 2000, pp. 99–102.
10. P. Grünbacher and C. Hofer, "Complementing XP with Requirements Negotiation," *Proc. 3rd Int'l Conf. Extreme Programming and Agile Processes in Software Eng.* (XP 02), Springer, 2002, pp. 105–108.
11. M. Lee, "Just-in-Time Requirements Analysis—The Engine That Drives the Planning Game," *Proc. 3rd Int'l Conf. Extreme Programming and Agile Processes in Software Eng.* (XP 02), Springer, 2002, pp. 138–141.
12. J. Araujo and J. C. Ribeiro, "Towards an Aspect-Oriented Agile Requirements Approach," *Proc. 8th Int'l Workshop Principles of Software Evolution*, IEEE Press, 2005, pp. 140–143.
13. O. Jepsen, "Time Constrained Requirement Engineering—the Cooperative Way," *Proc. 2002 Int'l Workshop Time-Constrained Requirements Eng.* (TCRE 02), 2002; www-di.inf.puc-rio.br/~julio/tcre-site/p5.pdf.

organizations didn't explicitly follow any specific "brand" of agile methods, they followed RE practices that were similar to those suggested by agile methods such as Extreme Programming (XP) and Scrum. In the second phase, we collected data from six organizations that used XP, Scrum, or both.

The participating organizations represent a rich mix of fields from healthcare to software development and consulting. We collected data through semistructured interviews, participant observations, and documentation review. In each organization, we interviewed a variety of stakeholders, including top management, product managers, quality assurance personnel, software developers, senior architects, and project managers. We also reviewed requirements documents such as story cards when available.

We conducted data analysis and data collection synergistically, as is common in qualitative research. Results from preliminary data analysis guided further data collection. Although most interviewees focused on current or recent project experiences, their responses included information on multiple projects from previous experience. So, each organization was the unit of data analysis.

To analyze the data, we used the grounded-theory method,[4] a well-established qualitative-research method. It lets you develop insights about a problem under investigation, without prior hypotheses. This approach is exploratory rather than confirmatory.

Data analysis involved open, axial, and selective coding.[4] In open coding, we identified groups of data and labeled them as agile RE practices, agile RE benefits, or agile RE challenges. We performed axial coding to uncover relationships among practices, benefits, and challenges. In selective coding, we identified larger patterns by systematically comparing the practices. We conducted additional interviews to gain clarification on some concepts. To generate insights from this analysis, we focused on similarities and differences in agile RE practices among the 16 organizations. Two coders separately coded the data, and we compared the

Study Participant Characteristics

Organization pseudonym	Industry and products	No. of employees interviewed	Organizational roles represented
Enco	Energy and communications. Offers forecasting tools.	3	VP of operations, project manager, and software developer
HealthCo	Healthcare and utilities. Offers an online service to help customers select health insurance and utility services.	6	President & CEO, VP of technology operations, director of marketing research, CIO, and developers
Venture	Across industries. Helps brick-and-mortar companies develop a Web presence.	4	Director, chief financial officer, chief operations officer, and developer
Entertain	Film and television industry. Offers high-tech indexing and search tools online.	4	Project manager, marketing specialist, senior Web developer, and quality assurance specialist
HuCap	Administration. Carries out human-resource administration for other companies online.	7	Project manager, architect, user interface designer, Web designers, and Web developers
TravelAssist	Transport and tourist industry. Offers online services.	6	Senior manager, project manager, quality assurance manager, lead developers, and Web developers
ManageRisk	Across several industries. Offers insurance online.	3	Human-resources manager, Internet site manager, and Internet site developer
Transport	Transportation and logistics industry. Offers services online.	6	CIO, senior manager, project manager, architect, senior developer, and 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, quality assurance manager, quality assurance specialist, and Web developers
HealthInfo	Healthcare information systems. Offers information systems solutions to hospitals, physicians' offices, and home healthcare providers.	2	Senior software engineers
SecurityInfo	Security software. Offers software for Internet security.	5	Software engineer, project lead, product manager, and quality assurance specialist
AgileConsult	Software consulting. Offers consulting services on agile software development.	2	Senior developer and project lead
EbizCo	Packaged software development. Offers e-business connections and transactions.	1	Senior software developer
FinCo	Online financial-transaction support. Offers online payments.	1	Software developer
NetCo	Network software consulting. Offers services on developing network systems and architectures.	2	General manager and senior software architect
BankSoft	Banking information systems. Offers software that handles financial transactions.	1	Senior software architect

results. Differences were resolved after detailed discussions. We then recoded the data to arrive at a set of practices common across these organizations.

The agile RE practices and their respective benefits and challenges we present reflect the perceptions and beliefs of the software developers who participated in the study.

Agile RE practices

Our study identified seven agile RE practices in the organizations.

Face-to-face communication over written specifications

According to the participants, agile RE aims to

effectively transfer ideas from the customer to the development team, rather than create extensive documentation. So, their agile RE practice prefers face-to-face communication over written specifications.

Most organizations shun formal documentation of specifications. Instead, they use simple techniques such as user stories to define high-level requirements. These short, abstract descriptions serve mainly as anchors for further discussions with customers. The developers discuss requirements in detail with the customers before and/or during development.

One exception is BankSoft, a company that develops banking-industry software and whose company policy mandates formal documentation. However, even for such security-critical applications, face-to-face communication with the customer is a primary source of requirements. The project team meets frequently with the product manager, who serves as a surrogate customer to discuss the requirements and alternative solutions. Formal documentation of requirements doesn't eliminate the need for frequent communication, because, as a BankSoft participant noted, "Everything is ambiguous; if you give me exactly what the customers want, they [the customers] are going to say, that's neat, [but] I want something different."

Benefits. All 16 organizations rely extensively on face-to-face communication between the team and the customers. The participants reported these benefits:

- Customers can steer the project in unanticipated directions, especially when their requirements evolve owing to changes in the environment or their own understanding of the software solution.
- Informal communication obviates the need for time-consuming documentation and approval processes, which are perceived as unnecessary, especially with evolving requirements.

Challenges. Several participants reported that this practice's effectiveness depends heavily on intensive interaction between customers and developers. For projects that can't achieve such high-quality interaction, this approach poses risks such as requirements that are inadequately developed or, worse still, wrong.

The effectiveness of communication between the customer and team depends on several factors, including customer availability, consensus among customer groups, and trust between the customer and the developers, especially during the project's early stages.

Many organizations reported that achieving on-site customer representation is difficult. In most of the projects we studied, product managers acted as surrogate customers. However, only two projects had a full-time, onsite product manager; the others had only part-time access.

When more than one customer group is involved, with each concerned about different aspects of the system, achieving consensus or compromise in the short development cycle is challenging. The development team must spend extra effort to integrate the requirements through negotiations with each group. For example, at NetCo, the project manager forced customers to physically participate in several meetings to discuss requirements, in order to achieve consensus.

Customers sometimes find it difficult to understand or trust the agile RE process. Many participants reported that establishing trust between the customer and developer, which is essential for agile RE, can be challenging. Customers familiar with a traditional development process might not understand or trust the agile RE process, which doesn't produce detailed requirements. One NetCo project included three customer representatives, but only one had a positive opinion of agile RE. In this project, the project manager suggested that the two customers who didn't have high confidence in agile methods weren't "good" customers in terms of their ability to provide relevant information and feedback.

Iterative requirements engineering

In 14 organizations, requirements aren't predefined; instead, they emerge during development. High-level RE occurs at the project's beginning. During this brief process, the development team acquires a high-level understanding of the application's critical features. Reasons for commencing development without spending much time on RE initially include high requirements volatility, incomplete knowledge of the technology used in development, and customers who can clearly define the requirements only when they see them ("I'll know it when I see it").

In most organizations, agile RE continues at each development cycle. At each cycle's start, the customer meets with the development team to provide detailed information on a set of features that must be implemented. During this process, requirements are discussed at a greater level of detail. Also, RE is often intertwined with design. This activity often results in a set of fine-grained requirements and a preliminary design, and sometimes even an implementation plan, none of which is specified formally.

Customers sometimes find it difficult to understand or trust the agile RE process.

Benefits. Iterative RE has two reported benefits.

First, it creates a more satisfactory relationship with the customer. Here's how a SecurityInfo customer compared his experience with agile RE and traditional RE: "I think the difference for me came in the quality of the software [and] the stability of the software. … I think the agile [RE] lent itself to … a very robust rich implementation of features … [for] the first time."

Second, requirements are clearer and more understandable because of the immediate access to customers and their involvement in the project when needed.

The participants suggested that iterative RE might be appropriate even in stable business environments where the changes often come from unforeseen technical issues, especially when adopting new technologies. For example, FinCo used a beta version of the .NET framework, and the evolving technology caused several changes to the requirements and the system design. In many organizations, the customers aren't clear at the outset about their requirements and are willing to explore the ways in which the evolving system can help their business goals. Flexible RE facilitates this joint discovery of potentially interesting solutions.

Challenges. Participants reported three major challenges.

The first is cost and schedule estimation. Because none of the organizations follow a formal RE phase, the initial estimation of project size typically is based on the known user stories. Many of these might be discarded, and many more get added during development. So, the original estimates must be adjusted (sometimes quite dramatically) during development, as happened with HuCap. Because the project scope is subject to constant change, creating accurate cost and schedule estimates for the entire project is difficult. Obtaining management support for such projects could be challenging.

The second challenge is minimal documentation. When a communication breakdown occurs owing to, for example, personnel turnover, rapid changes to requirements, unavailability of appropriate customer representatives, or the application's growing complexity, the lack of documentation might cause a variety of problems. These include, as one ServIT participant noted, the "inability to scale the software, evolve the application over time, and induct new members into the development team."

The third challenge is neglect of *nonfunctional requirements*, a major concern with iterative RE in agile development. Many participants acknowledged that NFRs are often ill defined and ignored during early development cycles. Customers often focus on core functionality and ignore NFRs such as scalability, maintainability, portability, safety, or performance. One common exception is the focus on ease of use, especially when the customers are intensely involved in providing constant feedback on the evolving system. Many participants, such as ServeIT and TravelAssist, suggested that the tendency to ignore critical issues such as security and performance early in the process results in major issues as the system matures and becomes ready for larger-scale deployment.

Requirement prioritization goes extreme

Agile development implements the highest-priority features early so that customers can realize the most business value. All the organizations prioritize their feature lists repeatedly during development as the customer's and the developer's understanding of the project evolves, particularly as requirements are added or modified.

The participants identified at least two important differences between traditional and agile RE in requirements prioritization. First, in traditional RE, requirements are typically prioritized once. In contrast, in the 16 organizations, agile RE involves prioritizing requirements in each development cycle. Prioritization often happens during the planning meetings at the beginning of each cycle. Moreover, requirements are prioritized together with other development tasks such as incorporating changes to existing functionality, bug fixes, and refactoring.

Second, in traditional RE, many factors drive requirements prioritization—for example, business value, risks, cost, and implementation dependencies. Customers identify the features that provide them the greatest benefit; developers identify technical risks, costs, or implementation difficulties. In contrast, agile RE practitioners uniformly reported that their prioritization is based predominantly on one factor—business value as the customer defines it.

Benefits. Because customers are very involved in the development process, they can provide business reasons for each requirement at any development cycle. Such a clear understanding of the customer's priorities helps the development team better meet customer needs. Even BankSoft, which uses formal requirement specifications, also benefits from frequent reprioritization of requirements because, according to one participant, "we were delivering high value every step of the way."

Also, in contrast to traditional development,

where achieving reprioritization is difficult, agile RE provides numerous opportunities for reprioritization.

Challenges. Using business value (often focused on time-to-market) as the only or primary criterion for requirements prioritization might cause major problems. For example, at FinCo, this approach resulted in an architecture that wasn't scalable. At Transport, it resulted in a system that couldn't accommodate requirements (such as security and efficiency) that might initially appear secondary to the customer but that become critical for operational success. Furthermore, some participants observed that continuous reprioritization, when not practiced with caution, leads to instability.

Managing requirements change through constant planning

Accommodating requirements changes during development is a way of tuning the system to better satisfy customer needs. Changes are easier to implement and cost less in agile development, a NetCo developer observed: "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."

Participants commonly reported two types of requirements changes: adding or dropping features, and changing already implemented features. At the end of each cycle, tests evaluate the implemented features. Customers provide feedback and can request major changes if their expectations aren't met. In the 16 organizations, this kind of change is relatively rare.

Such a low occurrence of major postdevelopment change is interesting because this ability is often touted as an important benefit of agile processes. The study participants believe that frequent communication between the developer and the customer during development obviates the need for changes after development. Before implementing a feature, the developer engages in detailed discussions with the customer to roughly understand what he or she needs. Also, the developer gets constant feedback from the customer as the features are implemented. Organizations that practice such intense interactions reported a low need for major changes to the delivered features.

Benefits. The early and constant validation of requirements largely minimizes the need for major changes. As an AgileConsult developer described, most of the change requests are "usually more a case of tweaks … spelling, little graphical things … for example, color, positioning."

So, the cost of addressing a change request decreases dramatically compared to traditional software development.

Challenges. In several organizations, such as FinCo and AgileConsult, the architecture the development team chose during the early cycles became inadequate as requirements changed. Redesign of the architecture added significantly to project cost.

Refactoring changes software's internal structure to make it easier to understand and cheaper to modify without changing its observable behavior. However, for most participants, the need for refactoring isn't always obvious, and the ability to refactor software depends on many factors, such as the developers' experience and schedule pressure. Moreover, some participants reported that refactoring, as an ongoing activity to improve the design, often doesn't completely address the problem of inadequate or inappropriate architecture. Occasionally, the only alternative is to throw away the code and rewrite entire modules. One AgileConsult developer reported that, because of this problem, he had to rewrite large application modules (200–330 KLOC) about five times.
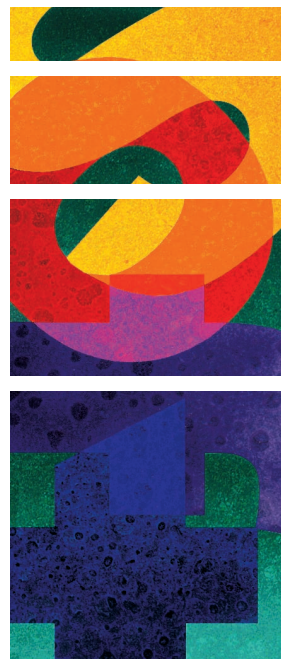
Prototyping

Many organizations, such as ServeIT, HuCap, Transport, and Venture, develop a prioritized list of features to settle requirements specification quickly. According to one ServeIT participant, "This helps reduce the margin of error. Piloting applications and releasing them to end users in iterative fashion are other useful practices."

To a certain extent, the production software itself can be a form of operational prototype, a refinement of the code created for experimentation with required features. In several organizations, the rush to market encourages a tendency to deploy these prototypes rather than wait for robust implementations. The ability to quickly deploy newer versions of the products on the Internet also contributes to this tendency.

Benefits. Instead of incurring the overhead involved in creating formal requirements documents, several organizations use prototyping to communicate with their customers to validate and refine requirements. Eleven organizations regularly use prototypes to obtain quick customer feedback on requirements.

Challenges. Some organizations are recognizing the risks of deploying prototypes in production mode. For example, at Entertain, maintaining or evolving prototypes is difficult and has caused problems with

features such as scalability, security, and robustness. Also, at TravelAssist, quick deployment of prototypes in the early stages has created unrealistic expectations among customers. They have been unwilling to accept longer development cycles that are necessary to develop more scalable and robust implementations as the product matures.

Test-driven development

TDD is an evolutionary approach in which developers create tests before writing new functional code. TDD treats writing tests as part of a requirements/design activity in which a test specifies the code's behavior. "You write code that talks about what the system's behavior should be. So you end up writing very explicit specifications and not 'tests,'" explained an AgileConsult developer.

Benefits. Many organizations use tests to capture complete requirements and design documentation that are linked to production code. This traceability makes incorporating changes easy, claimed an AgileConsult developer: "Having those tests allows 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."

Challenges. A major challenge to TDD's adoption is that developers aren't accustomed to writing tests before coding. Most developers in the study admitted that they don't consistently follow this practice because it demands a lot of discipline. Another challenge is that TDD requires a thorough understanding of the requirements and extensive collaboration between the developer and the customer, because it involves refining low-level specifications iteratively. Owing to these challenges, most organizations reported that they're unable to implement this practice.

Use review meetings and acceptance tests

Almost all the organizations use frequent review meetings for requirements validation. At the end of each development cycle, they hold a meeting with developers, customers, quality assurance personnel, management, and other stakeholders. During the meeting, the developers demonstrate the delivered features, and the customers and QA people ask questions and provide feedback. However, for many organizations, these review meetings cover only minor issues. As the SecurityInfo project manager described, "We basically get some minor feedback [from the review meetings]. ... The big thing is when, at the start of the iteration, I sit down with

the product manager [the surrogate customer] to talk about features. The PM sometimes brings up new things he found out as he was talking to more customers."

Acceptance tests that the customer develops, sometimes with help from the QA personnel, are another means for validation and verification. Some organizations treat these tests as part of requirements specification.

Benefits. In most organizations, the review meetings primarily provide progress reports to the customer and other stakeholders in the organization, even though the meetings' original purpose is to review the developed features and get feedback. The meeting's perceived benefits include the opportunities to ascertain whether the project is on target, to increase customer trust and confidence in the team, and to identify problems early during development. These meetings help considerably to obtain management support for the project by providing frequent updates on project status and progress to project sponsors.

Challenges. The participants suggested that their agile RE practice focuses more on requirements validation than traditional approaches. However, it doesn't address aspects of formal verification because there's no formal modeling of detailed requirements. Consistency checking or formal inspections seldom occur.

Although agile practices emphasize acceptance testing, several organizations find implementing such testing difficult owing to the difficulty of access to the customers who develop these tests. So, many organizations use QA personnel to help customers develop these tests.

A comparison of agile RE practices

To develop a detailed understanding of the agile RE practices, we determined the degree to which the 16 organizations reportedly followed them (see table 1). Most of the organizations rank high or medium for most of the practices. However, not all the organizations are encountering all the challenges of agile RE practices. Almost all the organizations reported that their most common challenges are the inability to gain access to the customer and obtaining consensus among various customer groups. TDD is the least-used practice (only six organizations adopt it) because, as we mentioned before, most developers aren't accustomed to the discipline it requires. Surprisingly, although prototyping is considered one of the most established practices, almost one-third of the organizations don't practice

Table 1
Agile requirements-engineering practices in 16 organizations

Adoption level	Practice						
	Face-to-face communication	Iterative RE	Extreme prioritization	Constant planning	Prototyping	Test-driven development	Reviews & tests
High	8	9	10	8	8	5	11
Medium	8	5	6	6	3	1	4
Low	0	2	0	2	0	0	1
None	0	0	0	0	5	10	0

it. While the literature on traditional development laments the inadequate attention paid to reviews and tests, the 16 organizations use them extensively in agile RE.

Our study reveals that agile RE differs from traditional RE in that it takes an iterative discovery approach. Agile development occurs in an environment where developing unambiguous and complete requirement specifications is impossible or even inappropriate. These fundamental differences have led to the set of agile RE practices we report here. The study participants identified the intensive communication between the developers and customers as the most important RE practice. Instead of following a formal procedure to produce a complete specification that accurately describes the system, agile RE is more dynamic and adaptive. As we mentioned before, agile RE processes aren't centralized in one phase before development; they're evenly spread throughout development.

Although agile RE practices provide benefits such as improved understanding of customer needs and the ability to adapt to the evolving needs of today's dynamic environment, they pose several distinct challenges. The study suggests that agile RE practices are neither panacea nor poison to the challenges intrinsic to RE. Development organizations, therefore, should carefully compare the costs and benefits of agile RE practices in their project environment. ⓈⓌ

References
1. B. Boehm, "Requirements That Handle Iᴋɪᴡɪsɪ, COTS, and Rapid Change," *Computer*, July 2000, pp. 99–102.
2. J. Erickson, K. Lyytinen, and K. Siau, "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research," *J. Database Management*, vol. 16, no. 4, 2005, pp. 88–99.
3. C.B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Trans. Software Eng.*, vol. 25, no. 4, 1999, pp. 557–572.
4. A. Strauss and J. Corbin, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, Sage Publications, 1990.
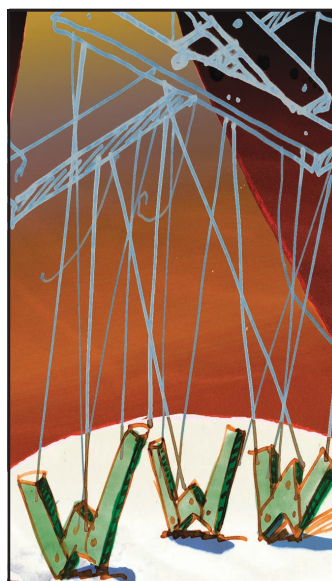
About the Authors

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 modeling and simulation. She received her PhD in computer information systems from Georgia State University. Contact her at the Dept. of Information Technology and Decision Sciences, Old Dominion Univ., Norfolk, VA 23529; lcao@odu.edu.

Balasubramaniam Ramesh is a professor of computer information systems at Georgia State University. He studies requirements engineering and traceability, agile software development, and knowledge management. He received his PhD in information systems from the Stern School of Business, New York University. He's a member of the IEEE, ACM, and the Association for Information Systems. Contact him at the Computer Information Systems Dept., Georgia State Univ., 35 Broad St., Atlanta, GA 30302; bramesh@gsu.edu.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.