Describing Agile Requirements Development and Communication using Complex Adaptive Systems Theory

Aaron Read
Information Systems and Qualitative Analysis, University of Nebraska at Omaha, Omaha, United States., aread@unomaha.edu

Robert Briggs
MIS Department, San Diego State University, San Diego, CA, United States., rbriggs@mail.sdsu.edu

Gert-Jan Vreede
Center for Collaboration Science, University of Nebraska at Omaha, Omaha, NE, United States., gdevreede@usf.edu

Describing Agile Requirements Development and Communication using Complex Adaptive Systems Theory

Aaron Read **Gert-Jan De Vreede**
Center for Collaboration Science, University of Nebraska
{aread,gdevreede}@mail.unomaha.edu

Robert O. Briggs
rbriggs@mail.sdsu.edu
MIS Department, San Diego State University

ABSTRACT

Agile software development methods help software development teams respond to changing requirements. Fundamental to this ability to respond to change is the manner in which requirements are communicated and developed. The question of agile requirements development is considered through the lens of Complex Adaptive Systems theory, a theory used to explain agility in software development teams. A case study of the communication and development of requirements in a software development team is reported, where the three dimensions of CAS Theory described by Vidgen and Colleagues (Vidgen and Wang 2009) are adapted to describe requirements communication and development practices in greater detail. We find that this focus on requirements practices can further explain increases in a software team's agility.

Keywords

Agile software development, Complex Adaptive Systems, User Stories

INTRODUCTION

Software development projects continue to suffer from low success rates (CHAOS 2009). Much of the difficulty with software projects focuses on the difficulty of developing requirements (Hofmann and Lehner 2001), especially when faced with requirements change. Evolving business needs are a reality of many organizations, which translate into evolving requirements (Hoorn et al. 2007). Many times requirements changes originate with changes in markets (Cusumano and Yoffie 1999). To respond more quickly to changing requirements, software development methods, such as those that fall under the umbrella of Agile (XP, Scrum, Crystal), have been adopted by an increasing number of software development teams (VersionOne 2010), to enable them to achieve faster times to market. At the heart of these Agile methods is the goal to enable the software team to increase its agility—to respond rapidly and effectively to changes in its environment (Conboy 2009) Teams that have this ability to respond quickly to changes in the environment are able to perform better in environments of change such as new product development (Pavlou and El Sawy 2006) and software development.

The agility of software development teams depends heavily on the manner in which requirements are communicated and developed. A key tenet of Agile software development emphasizes the use of face-to-face interaction and working software over comprehensive documentation as the means of communicating and specifying software requirements for developers (Beck et al. 2001). Agile methods such as XP and Scrum use one or two sentence "user stories" (Cohn 2004), as the sole means of documenting requirements. These user stories serve as reminders to hold conversations about how the software will work. These conversations, along with working software provide clear feedback mechanisms which allow stakeholders to quickly reveal communication gaps caused by a lack of shared assumptions (Maiden and Rugg 1996).

Because of the complexity of a software project, or other environmental characteristics, Agile teams may need to tailor the requirements communication and development practices suggested in Agile software development methods. If requirements are difficult are very complex, the team may adopt more structured methods of documenting requirements (Mathiassen et al. 2007). Diagrams can be used in Agile settings to develop and communicate understanding of complex requirements (Ambler 2004). Some form of documentation based analysis can be helpful or even necessary in reasoning about non-functional requirements (Turk and France. R 2005). Examples of combining Agile methods with more complex

methods of documenting requirements in industry have reportedly been highly successful (Armano and Marchesi 2000; Mohan et al. 2010).

While case studies describing the adaptations of Agile requirements communication and development practices have demonstrated the successfulness of the adaptation, there have not been any studies which have explained theoretically how requirements development and communication adaptation decisions have successfully resulted in an increase in agility. Other case studies focus on high-level practices such as the continuous engagement of customers and using a flexible architecture (Mohan et al. 2010) or the agility of the team in terms of broader Agile project management practices (Vidgen and Wang 2009). While such findings are highly useful, they do not capture the rich, experiential attributes of requirements communication and development practices. Other studies in Agile software development settings have used descriptive frameworks such as distributed cognition which describes interactions between individuals and artifacts in great detail (Abdullah et al. 2010), but do not describe how these contribute to agility.

We therefore report on a case study exploring and discussing several critical work practice incidents related to requirements communication and development which could shed light on the agility of the work practices. We assess the extent to which CAS Theory explains why changes in requirements communication and development work practices lead to increased agility. CAS theory is useful for explaining how decisions made at the micro or individual level lead to macro level effects (Nan 2011), such as the increased agility of a team. We discuss these work practices in terms of dimensions adapted from of Complex Adaptive Systems Theory set forth by Vigden and Colleagues (Vidgen and Wang 2009) to focus on requirements communication and development practices.

BACKGROUND

The effect of requirements communication and development practices on agility

Requirements communication and development practices play a critical role in Agile Software Development methods. These practices are consistently mentioned. The Agile manifesto directs teams to emphasize working software over documentation as a deliverable (Beck et al. 2001). Commercial Agile Methods packages, such as XP and Scrum include specific directions for the communication and development of requirements (e.g. user stories, acceptance tests, regression testing scripts) (Abrahamsson et al. 2003).

Recent theoretical developments have been able to explain why these practices play such a critical role in software development agility. Several researchers agree that some notion of effectiveness and efficiency in responsiveness to change (changes in requirements) are at the core of a team's ability to be agile (Lee and Xia 2010). Harris, Collins, and Hevner (2009) demonstrate that the positive effect of Agile Software development largely due to such team's working with a flexible, bounded description of the software as opposed to a detailed, fixed description. The bounded scope of the requirements gives the software team direction, while the minimal detail of specifications allows the team to respond quickly to feedback to changing needs. The study of Vidgen and Wang (Vidgen and Wang 2009) also found that the ability of a software development team to respond quickly to requirements change was dependent on requirements development practices—user stories and planning sessions were key to enabling this change.

The need for further exploration of requirements development and communication practices

There are practical and theoretical reasons why such a general description of the requirements communication and development practices suitable for agility are not adequate. From a practical and methodological perspective, the desire to increase the agility of projects with more complex requirements has been of interest for several years (Boehm and Turner 2003; Conboy and Fitzgerald 2010), with the goal of reducing the risks originating with project complexity. Ambler (Ambler 2004) has also coined the term "Agile Modeling" to describe the use of more formal requirements modeling techniques in Agile settings in order to increase communication and reasoning effectiveness without sacrificing efficiency.

There are also theoretical reasons for an exploration of requirements related practices. Aside from the ease with which they can be written, there has been little theoretical justification for the use of user stories in agile software development settings. Given the rich research which considers optimizing the structure of requirements for a given task (Shaft and Vessey 2006)and for the knowledge of the individuals performing tasks with the requirements (Khatri et al. 2006), such an exploration of requirements related practices in Agile settings is likely to provide theoretically meaningful results.

CAS SYSTEMS THEORY ORIGIN AND APPLICATION TO REQUIREMENTS PRACTICES

Complex Adaptive Systems Theory

Complex Adaptive Systems Theory explains how macro effects such as increased agility from three essential elements: 1) agents, 2) the interactions between them, and 3) the environment they interact with (Nan 2011). There is not one agreed upon paradigm within the CAS theory, even when describing agility in software teams (e.g. Mohan et al. 2010; Vidgen and Wang 2009). In explaining agility, the CAS principles included when explaining software development agility revolve around the ability of the CAS to adapt to its environment at an appropriate rate of change, through a response to feedback. We adapt the three principles of CAS set forth in (Vidgen and Wang 2009) because they include a greater focus on the extent to which the software team members are able to perform their own work and work with others. The reason for this focus is discussed as the adapted dimensions are described below.

Three dimensions of Complex Adaptive Systems

The adaptation of the three dimensions of CAS as described in Vidgen and Wang (2009) revolve around the new focus on not only the rate of change of requirements, but also the nature of the problem domain that the requirements represent. For example, in the study of Mohan and colleagues (Mohan et al. 2010), a commonality analysis of common features in an envisioned family of software products (software product line) allowed software teams to leverage the advantages of this software development paradigm in recognizing opportunities for software reuse within an agile setting.

Seek to match rate and nature of change in the environment

A CAS evolves in reaction to its external environment in order to survive. In a software development team, the way that the customer's problem and how it is satisfied can be considered out of the team's control (generally) and external to the team. As a team discovers the nature of its customer's problem, it needs to be able to assess whether or not its method of communicating and developing requirements is appropriate for the customer's problem. Careful design of the requirements communication and development practice which considers the best way to represent, communicate, and track changes in requirements represent ways a team may evolve in response to changes in the external environment.

Support self-organization in interactions between agents

The agents in a CAS (team members) are best able to respond to changes in the external environment when they are able to evolve internally with respect to the actions and needs of the individual agents. They participate in decisions and have an awareness of each other's actions which allows one team member's change to the environment to influence the entire system. This must be reflected in choices about requirements communication and development methods so that a method chosen to communicate requirements does not unnecessarily impede any member of the team. As understood from (Shaft and Vessey 2006) and (Khatri et al. 2006), the structure of requirements communication may be suitable for one task (e.g. finding commonalities in software) but may not be suitable for another, or for someone with a different knowledge set.

Balance exploration and exploitation

A team should acknowledge best practices while enabling the discovery of new practices. In terms of requirements communication and development efforts, the need to invest effort in documentation to enable the team to take advantage of consistency, while on the other hand being able to recognize flaws with the current requirements communication and development practice and areas where documentation may not apply.

METHOD

An exploratory case study was performed to describe events and artifacts experienced and developed by members of the project team who lived through several of the issues and subsequent improvements to the agility of the requirements communication and development processes. The goal of the case study is both illustrate and assess the extent to which the dimensions of a Complex Adaptive System, as presented in (Vidgen and Wang 2009) describe changes in a software development team's requirements communication and development work practices towards increased agility. The data for the analysis are a collection of critical instances where the current work practice was not working and the corresponding work practices developed or modified by the team as part of an evolution from the user story method described in (Cohn 2004) and other Agile Software Development practices (Abrahamsson et al. 2003). Data also includes some general observations about documentation work practices used by the team. These data are collected from field notes kept by the authors who participated on the project. Due to space constraints, we present some of the more influential changes in user story documentation and documentation process, which illustrate the explanatory power of CAS theory in explaining how the improvements enabled agility. There are many more examples which could be given.

Background of Case Study

The software development team under study was tasked with creating a new rapid development environment for collaborative software applications called *ActionCenters*. The application would be based off of experiences with an earlier prototype, with significant design effort devoted to enhancing the UI. The team had a budget of $3,600,000 USD which was anticipated to be only sufficient to build the core functionality of the system. The team consisted of the Product Owner (PO), who had the knowledge of the desired functionality of the application, the User Interface/User Experience designer (UI/UX), a System Architect (SA), a team of six developers, a Story Master (SM) who helped the customer in writing and managing user stories, and team of three testers. The team employed the Scrum methodology with the project with month long iterations. User stories were initially written according to the description given by Cohn (Cohn 2004)—they were one sentence long, represented functionality that could be developed in a sprint, given a priority and an estimation.

Case study analysis

We organize the data for the analysis according to its fit with one of the three dimensions of a Complex Adaptive System, and requirements related software development practices within Table 1. In the table, the critical incident is written in *italics* with the resulting change in practice written underneath. All non-development activities where requirements are used by members of the software development team are included: specification design, sprint planning, and software testing. The work practice changes are classified as follows:

Seek to Match External Rate and Complexity of Change: Work practice included any structural changes which reflected structures within the problem domain.

Support Self-Organization: Work practice changes which reflected changes in team practices that lead to an increase in the ability of team members to perform team and individual tasks.

Balance Exploration and Exploitation: Work practice changes where decisions to limit or increase the extent to which a work practice must be followed were made.

A narrative summary of the evolution of the team with respect to each of the three dimensions follows the description of initial work practices below.

Initial Work Practices

The initial work practices are summarized as follows:

- Initial development of a backlog of user stories at the beginning of the project
- User stories are one sentence long descriptions of how the software works
- User stories are discussed in meetings prior to a monthly sprint planning session. Acceptance tests are developed prior the sprint planning session.
- Decisions are finalized about requirements for the sprint as user stories are estimated and placed into work for the current sprint.

Evolving to match external rate and complexity of change

A CAS survives by being able to evolve both the rate and complexity of the changes in its environment. The major source of the complexity for the team was the requirements—both the rate at which requirements evolved and the level of detail in the requirements. Since the goal of the project centered around evolving the UI of an existing prototype, the needs of the PO did not change, aside from reprioritizations of needs as the project progressed. However, the design specifications of the software, which are often difficult to couple from customer need or requirements, changed every iteration of the project as the PO reflected on built software and spent more time discussing designs with the UI/UX designer, often making old user stories in the backlog obsolete. The team responded to the rate of change of requirements by coordinating design so that a design sprint occurred just before a development sprint.

The requirements for the software also exhibited patterns at times. Interface functionality was very detailed and very interrelated. Keyboard shortcuts, mouse clicks, and menu items often performed the same functionality on multiple objects.

EPIC: Open and Close Element editors
STORY: Open Element Editor
In [a CASE Editor], a Collaboration Engineer [opens] the element editor for an [object] to configure the features of that [object]. The element editor for that [object] appears on a new tab in the CACE editor pane. Fields with values set appear with those values. Fields with no values set appear empty.
BUSINESS RULES:
The first time an element editor for an [object] is opened, default values will be set for all fields
END RULES
END STORY
STORY: Close Element Editor
In the CACE, A Collaboration Engineer [closes] an [object] element editor to hide the configurable features of that [object] to reduce visual load. The element editor for the object closes. The values of all fields in the element editor are saved to the server.
BUSINESS RULES
A user cannot not close an element editor until the all its fields have displayed
END RULES
END STORY
PARAMETERS:
[a CACE Editor] Explorer Tree; ActionCenter Builder; Screen Editor; Tool Editor
[closes] clicks the x on the element editor tab; right clicks element editor tab and selects close; clicks the close button on the element editor
[object] Project; Phase; Activity; Role; Screen; Tool; Component; Control
[opens] Double-clicks; Right-clicks and selects "Edit"; Clicks the File menu and selects "Edit"
END PARAMETERS
END EPIC

Many actions performed by users had to be constrained by the same set of business rules. These patterns can be seen in the example of a HyperEpic in Figure 1. The HyperEpic and HyperStory formats were developed in response to the repeating patterns found in the software. The use of these formats facilitated the elicitation of requirements, and assured that functionality was consistent. Patterns in the requirements were also used during testing by developing key words based off of the parameters in HyperEpics.

FIGURE 1. Example of a small HyperEpic

Support Self-organization

The team evolved to support self-organization by reordering the process of communicating and developing requirements in a way that did not restrict any member of the team unnecessarily. The documentation of user stories during design meetings was also streamlined from user stories to story titles to make the meetings more effective and efficient. Finally, requirements were communicated briefly through HyperEpics and Hyperstories, then reverted back to being communicated principally through user stories with small annotated details, since such communication was easier to digest.

Balance exploitation and exploration

Because the application was developed as part of a research grant, the team had a natural tendency to explore new ways of documenting requirements. As illustrated by the examples above, these efforts were often rewarded by increases in efficiency and effectiveness. However, it was also useful for the team to stick to and codify documentation practices. The effectiveness of Story Titles, short names representing agreements to design decisions recorded during design meetings, came in part from the ability of participants in the design meeting to recognize when an agreement had taken place. Adherence to the process of recording story titles allowed participants in design sessions to develop the habit of recognizing when decisions about design had been reached.

	Match or Exceed Complexity	Support Self-Organization	Balance Exploration with Exploitation
Design	-*User stories become obsolete quickly due to rapid design changes. The backlog contains many obsolete user stories*. Designing by sprint becomes more systematic and coordinated with the coding sprints. Design sprints are coordinated so that the user stories that have received the most design attention are the ones that are the most likely to be developed in the upcoming sprint. -*Much of the functionality contains similar patterns. Hyperstories and HyperEpics which were subsequently developed.* HyperEpics and Hyper Stories take advantage of structure in the domain, by encouraging and facilitating thinking about the many repeated elements found in user stories (e.g. CRUD commands for the several objects in the domain).	-*User stories created before design sessions with the UI/UX expert unnecessarily constrain his designs.* User stories become the output of design decisions instead of the input to design sessions. -*User Story Capture interrupts meeting flow*. User Story Titles, one or two word phrases are used to capture points of agreement in meeting. Details about what was agreed upon are recorded in a format which facilitates recall immediately after the meeting.	-*HyperEpics and HyperStories are only used occasionally*. Hyperstories were useful to write only in situations where their structure provided to be an advantage. Especially user stories which resulted from the PO's feedback upon viewing the software were not written as HyperEpics or HyperStories. -*Mutually understood terms for commonly used concepts were helpful when maintain throughout the project.* Terms such as "Story Title" were useful to the team to understand what was being communicated about a design—in the case of story titles, they represented agreements about the design which could be further fleshed out into details.
Spring Planning	-*Some user stories contained complex business rules that needed to be connected to a single user story*. Sometimes when user stories contained complex business rules, the business rules were written up in a document and connected to the user story. Reading of the business rules document before the sprint planning meeting was required.	-*Developers were surprised by details in the user story acceptance tests which dramatically altered the amount of time the user story would take to write.* All though acceptance tests were written before sprint planning meetings, they were not discussed with developers during the sprint planning meetings, and were not as easily viewable as user stories. The team responded by having the SM review each story for complexity and likely effort so that such surprises could be avoided. -*HyperEpics and Hyperstories were not read by developers during meetings or throughout the sprint to obtain further details about a user story*. After the PO noticed that the HyperEpics were not being read, all important information from a HyperEpic, pertaining to a particular user story was put in a place easily viewable with the user story they explained.	-*The team noticed frequent obsoletion of user stories and frequent user story duplications in the backlog*. Instead of maintaining the backlog regularly, the backlog was periodically maintained (once every three months or less) to remove redundant, obsolete, or completed user stories.
Testing	-*Regression Testing Contains Several Repeating elements that needed to be tested systematically.* Acceptance testing takes advantage of structuring in the domain in a similar manner to HyperEpics. Regression tests were searchable by recurring objects, actions, and input methods, for example.	-*Acceptance tests written by Story Master were not adequate/understandable for testing purposes.* Acceptance Test writing shifted from the story master to test lead. This minimized errors in understanding of the acceptance tests. -*Acceptance testing was not visible enough to the SM to allow him to advice and monitor the testing effort.* Use of a single tool (VersionOne) for acceptance test writing, regression test writing, and defect tracking, along with the use of user story project management software already in place became the practice.	-*Automated regression tests were not providing feedback for the team fast enough to inform the team of the status of defects.* Automated regression test writing was abandoned and manual testing, with a focus on exploratory testing became the standard practice of the team. This was more effective for a couple reasons: First, automated testing scripts used by the software did not handle many of the ways the user interacted with the software (especially drag and drop functions). Secondly exploratory testing allows a tester to use intuition to use the system in ways that are more likely to reveal defective behavior.

Table 1: Critical Events and Team Efforts to Increase Agility, Classified by Complex Adaptive System Dimension and Software Requirements Practice

DISCUSSION

The adaptation of the three dimensions of CAS Theory as described in (Vidgen and Wang 2009) to emphasize both the rate of change of requirements and the characteristics of requirements provided for the discovery of additional opportunities to be Agile. Adapting the format of requirements elicitation to reflect structures in the problem domain, as was done using the HyperEpic, HyperStory to capture requirements was a prominent example of the importance of better understanding the environment which a CAS must adapt to. Like the optimal match between tasks and task representations in cognitive fit theory (Shaft and Vessey 2006), there is also likely an optimal match between the way requirements are represented, and the character (including complexity) of the requirements.

The CAS cannot simply adapt requirements work practices to such changes in the environment without considering the two other dimensions, however. There seems to be a natural tension between the three different dimensions, that should be recognized explicitly. As an example of this tension the current case study, it appears that one cannot match the rate and complexity of change in a way that undermines self-organization. The practice of writing and communicating requirements as HyperEpics and HyperStories increased agility for the customer, but made the process of understanding requirements more difficult for developers—making them less autonomous in their approach. We have also emphasized matching requirements complexity here, something that is suggested by other authors in approaching agility (Boehm and Turner 2003). This concept is not stressed in Vidgen and Wang's dimension (match rate of change). Balancing exploitation and exploration cannot be neglected in work practices—the software development team must invest in thinking which allows them to leverage characteristics of the problem domain only when it serves to be an advantage.

CONCLUSION

Using CAS Theory, we have illustrated how requirements development and communications decisions can influence the agility—the effectiveness and efficiency in response to change—of a software development team. We have shown that CAS theory can explain the rationale of these decisions. The theory is useful for aiding practitioners and researchers seeking to understand the importance of characteristics of requirements communication and development methods, but that more understanding is needed to evaluate these practices Future research should assess work practice design decisions made by other software teams in other settings to assess the generalizability of our findings presented here.

REFERENCES

1. Abdullah, B., Sharp, H., and Honiden, S. 2010. "A Method of Analysis to Uncover Artefact-Communication Relationships," *FLAIRS Conference 2010*, pp. 349-354.
2. Abrahamsson, P., Warsta, J., Siponen, M.T., and Ronkainen, J. 2003. "New Directions on Agile Methods: A Comparative Analysis." Portland, Oregon: IEEE Computer Society, pp. 244-254
3. Ambler, S. 2004. *The Object Primer: Agile Model-Driven Development with Uml 2.0*. Cambridge University Press.
4. Armano, G., and Marchesi, M. 2000. "A Rapid Development Process with Uml," *SIGAPP Applied Computing Review* (8:1), pp 4-11.
5. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Jeffries, R.E., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. 2001. "Manifesto for Agile Software Development." (avialable at www.agilemanifesto.org)
6. Boehm, B., and Turner, R. 2003. "Using Risk to Balance Agile and Plan-Driven Methods," *Computer* (36:6), pp 57-66.
7. CHAOS. 2009. "Chaos Report Summary." 2009, from http://www1.standishgroup.com/newsroom/chaos_2009.php
8. Cohn, M. 2004. *User Stories Applied: For Agile Software Development*. Addison Wesley Longman Publishing Co., Inc. .
9. Conboy, K. 2009. "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development," *Information Systems Research* (20:3), pp 329-354.
10. Conboy, K., and Fitzgerald, B. 2010. "Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of Xp Expert Opinion," *ACM Transactions on Software Engineering Methodology* (20:1), pp 1-30.
11. Cusumano, M.A., and Yoffie, D.B. 1999. "Software Development on Internet Time," *Computer* (32:10), pp 60-69.
12. Harris, M., L., Collins, R.W., and Hevner, A.R. 2009. "Control of Flexible Software Development under Uncertainty," *Information Systems Research* (20:3), pp 400-419.
13. Hofmann, H.F., and Lehner, F. 2001. "Requirements Engineering as a Success Factor in Software Projects," *IEEE Software* (18:4), pp 58-66.
14. Hoorn, J.F., Konijn, E.A., van Vliet, H., and van der Veer, G. 2007. "Requirements Change: Fears Dictate the Must Haves; Desires the Won't Haves," *Journal of Systems and Software* (80:3), pp 328-355.

15. Khatri, V., Vessey, I., Ramesh, V., Clay, P., and Park, S.-J. 2006. "Understanding Conceptual Schemas: Exploring the Role of Application and Is Domain Knowledge," *Information Systems Research* (17:1), pp 81-99.
16. Lee, G., and Xia, W. 2010. "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility," *Management Information Systems Quarterly* (34:1), pp 87-114.
17. Maiden, N.A.M., and Rugg, G. 1996. "Acre: Selecting Methods for Requirements Acquisition," *Software Engineering Journal* (11:3), pp 183-192.
18. Mathiassen, L., Tuunanen, T., Saarinen, T., and Rossi, M. 2007. "A Contingency Model for Requirements Development," *Journal of the Association for Information Systems* (8:11), pp 569-597.
19. Mohan, K., Ramesh, B., and Sugumaran, V. 2010. "Integrating Software Product Line Engineering and Agile Development," *IEEE Software* (27:3), pp 48-55.
20. Nan, N. 2011. "Capturing Bottom-up Information Technology Use Processes: A Complex Adaptive Systems Model," *Management Information Systems Quarterly* (35:2), pp 505-532.
21. Pavlou, P.A., and El Sawy, O.A. 2006. "From It Leveraging Competence to Competitive Advantage in Turbulent Environments: The Case of New Product Development," *Information Systems Research* (17:3), pp 198-227.
22. Shaft, T.M., and Vessey, I. 2006. "The Role of Cognitive Fit in the Relationship between Software Comprehension and Modification," *Management Information Systems Quarterly* (30:1), Mar, pp 29-55.
23. Turk, D., and France. R, R. 2005. "Assumptions Underlying Agile Software-Development Processes," *Journal of Database Management* (16:4), pp 62-87.
24. VersionOne. 2010. "State of Agile Survey." VersionOne.
25. Vidgen, R., and Wang, X. 2009. "Coevolving Systems and the Organization of Agile Software Development," *Information Systems Research* (20:3), pp 355-376.