Self-Organizing Roles on Agile Software Development Teams

Rashina Hoda, *Member*, *IEEE*, James Noble, *Member*, *IEEE*, and Stuart Marshall, *Member*, *IEEE*

Abstract—Self-organizing teams have been recognized and studied in various forms—as autonomous groups in socio-technical systems, enablers of organizational theories, agents of knowledge management, and as examples of complex-adaptive systems. Over the last decade, self-organizing teams have taken center stage in software engineering when they were incorporated as a hallmark of Agile methods. Despite the long and rich history of self-organizing teams and their recent popularity with Agile methods, there has been little research on the topic within software wngineering. Particularly, there is a dearth of research on how Agile teams organize themselves in practice. Through a Grounded Theory research involving 58 Agile practitioners from 23 software organizations in New Zealand and India over a period of four years, we identified informal, implicit, transient, and spontaneous roles that make Agile teams self-organizing. These roles—Mentor, Coordinator, Translator, Champion, Promoter, and Terminator—are focused toward providing initial guidance and encouraging continued adherence to Agile methods, effectively managing customer expectations and coordinating customer collaboration, securing and sustaining senior management support, and identifying and removing team members threatening the self-organizing ability of the team. Understanding these roles will help software development teams and their managers better comprehend and execute their roles and responsibilities as a self-organizing team.

Index Terms—Self-organizing, team roles, software engineering, Agile software development, grounded theory

1 Introduction

Self-organizing teams have been recognized and studied in various forms—as autonomous groups in socio-technical systems as early as in the 1950s, as enablers of holographic organizations in organizational theory and as agents of knowledge creation and management around the 1980s, and as examples of entities exhibiting spontaneous order in Complex Adaptive Systems (CAS) in the 1990s. More recently, with the rise of Agile methods in the late 1990s and early 2000s, self-organizing teams took center stage in the software engineering arena when they were incorporated as a hallmark of Agile software development [1].

Self-organizing teams are at the heart of Agile software development [2], [3], [1], [4], [5], [6]. Self-organizing Agile teams are composed of "*individuals [that] manage their own workload, shift work among themselves based on need and best fit, and participate in team decision making*" [7]. Self-organizing teams must have common focus, mutual trust, respect, and the ability to organize repeatedly to meet new challenges [3]. The scrum method specifically mentions self-organizing Agile teams and the concept of "*empowered*" teams has recently been added to XP [8].

Self-organizing teams are not only seen as enabling Agile engineering practices, but also as capturing the spirit of Agile values and principles, which focus on human and social aspects of software engineering. Self-organizing teams is one of the principles behind the Agile Manifesto and have been identified as one of the critical success factors of Agile projects [9], [2], [1].

Despite the multifaceted history of self-organizing teams and their recent popularity with Agile methods, there has been little research on the topic within software engineering. In particular, while there an increasing interest in research on Agile software development teams in general, there is a dearth of research on the specific topic of self-organization in Agile teams and how Agile teams organize themselves in practice. This paper presents the results of a qualitative Grounded Theory (GT) study involving 58 Agile practitioners from 23 different software organizations in New Zealand and India conducted over a period of four years. As a part of the study, semistructured, face-to-face interviews were conducted with Agile practitioners in the New Zealand and Indian software industries, using open-ended questions. The interviews were supplemented by observations of workplaces and various Agile activities and artifacts. The data were analyzed using Grounded Theory's constant comparison method via open, selective, and theoretical coding procedures to find the most important themes in the data across a majority of the participants.

This research resulted in a grounded theory of self-organizing Agile teams which is comprised of the three main themes that emerged from the study, namely, *self-organizing roles*, *self-organizing practices*, and *critical factors influencing self-organizing teams*. The two themes—self-organizing practices and critical factors influencing self-organizing teams—have been described elsewhere [10], [11], [12], [13] and are not covered in this paper.

This journal paper presents the most significant theme that emerged from the research—the six informal, implicit, transient, and spontaneous roles on Agile software development teams that enable self-organization. These roles are:

- *R. Hoda is with the Department of Electrical and Computer Engineering, The University of Auckland, Private Bag 92019, Auckland Mail Centre, Auckland 1142, New Zealand. E-mail: r.hoda@auckland.ac.nz.*
- *J. Noble and S. Marshall are with the School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand. E-mail: {kjx, stuart}@ecs.vuw.ac.nz.*

Manuscript received 9 June 2011; revised 28 Mar. 2012; accepted 8 Apr. 2012; published online 4 May 2012.
Recommended for acceptance by L. Williams.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2011-06-0183.
Digital Object Identifier no. 10.1109/TSE.2012.30.

1. *Mentor*, who guides and supports the team initially, helps them become confident in their use of Agile methods, ensures continued adherence to Agile methods, and encourages the development of self-organizing practices in the team.
2. *Coordinator*, who acts as a representative of the team to manage customer expectations and coordinate customer collaboration with the team.
3. *Translator*, who understands and translates between the business language used by customers and the technical terminology used by the team to improve communication between the two.
4. *Champion*, who champions the Agile cause with the senior management (SM) within their organization in order to gain support for the self-organizing Agile team.
5. *Promoter*, who promotes Agile with customers and attempts to secure their involvement and collaboration to support the efficient functioning of the self-organizing Agile team.
6. *Terminator*, who identifies team members threatening the proper functioning and productivity of the self-organizing Agile team and engages senior management support in removing such members from the team. We have selected quotations drawn from our interviews that shed particular light on the concepts.

Section 2 of this paper covers a detailed literature review on software development teams and self-organizing teams from multiple perspectives—such as socio-technical, organizational theory, knowledge management, complex-adaptive systems, and Agile software development.

Section 3 presents our research method, Grounded Theory. We dedicated a separate journal article entirely to the description of our research method and its application, submitted originally as a part of a special section on qualitative research methods in software engineering [14]. Section 3 of this paper presents the basics of Grounded Theory and refers to our other dedicated publication on GT for further details.

Section 4 presents the self-organizing Agile team roles in full detail. A preliminary description of these roles was presented in our earlier conference paper [15]. This paper expands on the conference paper in the following ways: While the conference paper presented the roles as they were emerging from the early stages of our study based on 24 participants from relatively new Agile teams, this journal paper provides an in-depth description of the self-organizing roles on Agile teams as they finally emerged out of the completed four-year study involving 58 Agile practitioners from 23 software organizations. It captures the evolution of the roles as we moved on to study 34 more Agile practitioners focusing on more mature Agile teams toward the second half of our research.

Section 5 presents a discussion of our findings—the self-organizing roles—in light of the related literature such as Belbin's nine behavior-based team roles and Ancona and Caldwell's five boundary-spanning roles [16], [17]. Such insight into related work appeared after a major literature review was conducted toward the end of the research to consolidate our theory of self-organizing Agile software development teams with related theories outside the field of software engineering. We also present limitations and threats to validity in this section. Future work suggested in this paper is derived from the discovery of self-organizing roles being mirrored at an organization-wide level in two highly mature Agile companies studied toward the end of the study. Finally, the paper concludes in Section 6.

2 Literature Review

2.1 Agile Software Development

Agile software development methods emerged in the late 1990s [18]. Agile methods follow an iterative and incremental style of development where collaborative self-organizing teams dynamically adjust to changing customer requirements [19], [4]. The developers (Dev) of some of these methods collaboratively wrote the Agile Manifesto [1] and used "Agile" as an umbrella term for several iterative and incremental methods. The Agile Manifesto values:

> *individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan. That is, while there is value in the items on the right, we value the items on the left more.*

The principles behind the Agile Manifesto include fast, frequent, consistent, and continuous delivery of working software; responding to changing requirements; encouraging effective communication; and motivated and well-supported self-organizing teams.

Agile methods were developed as a response to the weaknesses of traditional software development models [20]. Agile methods improve over the traditional software development models by accommodating changes through an iterative and incremental style of development, allowing each iteration to focus on a small set of functionalities prioritized by the customer. Agile methods encourage continuous customer involvement and feedback, and allow the customer to prioritize the features they want developed first.

Some flavors of Agile methods include: *Dynamic Software Development Method* (DSDM), referred to as the first Agile method [21]; *Crystal*, a family of methodologies consisting of a number of methods, and principles for customizing them for particular projects [22], [23]; *Feature Driven Development* (FDD), which focuses on features-based division of work [24]; and *Adaptive Software Development* (ASD), which focuses on concepts and culture, and creating emergent order "*out of chaos*" [22], [25]. The most widely adopted Agile methods include *Scrum* and *XP* [26].

Scrum was developed by Jeff Sutherland and formalized by Ken Schwaber [27]. Scrum derives its roots from Takeuchi and Nonaka's paper in 1986 "The New New Product Development Game" in the *Harvard Business Review* [28]. Scrum is characterized by sprints work cycles, typically two to four weeks [29]. During each sprint, self-organizing teams pick tasks from a prioritized list of customer requirements so that the features that are developed first are of the highest value to the customer. At the end of each sprint, a potentially shippable product is delivered.

eXtreme Programming (XP) was developed by Kent Beck and Ward Cunningham, with support from Ron Jeffries and

Martin Fowler [30]. XP is defined as "*a light weight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements*" [30]. XP was developed to address and solve some of the classic problems in software development such as schedule slips, canceled projects, inability to solve business problem, and richness of features, with little business values. By advocating short release cycles, XP tries to limit the scope of schedule slips. XP asks customers to select the smallest release that makes maximum business value. In this way, XP tries to help reduce the amount of things that can go wrong at production, thereby reducing the risk of the project being canceled. XP requires the customer to be a part of the team and provide rapid feedback so that the business values are not misunderstood while developing features. XP insists on only highest priority features being implemented and tries to reduce the bulk of features with little or no business value.

Most XP practices are focused around development activities at the team level: in contrast, Scrum focuses more on project management [31], [32].

2.2 Traditional Software Development Teams

Traditional software development is characterized by manager-led teams, organized in a hierarchical structure with multiple layers of authority [33]. Management in traditional teams is typically command and control style [34]. Roles on traditional teams are based around functional tasks reflected by their organizational roles, such as programmers responsible for programming, testers responsible for testing, analyst responsible for requirements analysis, etc. Work is delegated to team members by their managers. Practices of traditional teams include documentation, specifications, and planning [35], [34]. There are indirect lines of communication across the different layers of the organizational hierarchy. Members in hierarchical team structures were commonly lacking in empowerment and visibility of the overall project [33].

The Chief Programmer team and the Surgical team are examples of hierarchical teams designed to tackle large software systems development [36], [37]. The Chief Programmer team consists of the Chief Programmer—responsible for the team—the Backup Programmer, and the Librarian. The Surgical team was an extension of the Chief Programmer team, with as many as 10 members [36]. In addition to the three roles in a Chief Programmer team, the Surgical team includes an editor—responsible for documentation; an administrator—responsible for tedious, nonproduct-related tasks; a couple of secretaries—responsible for helping the editor and the administrator; a toolsmith—an expert in the tools and operating system; a tester—responsible for functional testing; and a language lawyer—an expert in the language being used on the project [33].

2.3 Agile Software Development Teams

A hallmark of Agile software development is its focus on people and social interactions. Agile teams are meant to be democratic teams—where all members are considered peers at the same level, without a strict hierarchy in practice. Team members are empowered with collective decision-making and cross-functional skills, which increases their ability to self-organize [34]. Management in Agile teams is meant to be more facilitative and coordinating [34]. Smaller teams are better suited to democratic structures than larger teams [33]. This is one of the reasons that Agile teams work best in smaller numbers [38], [34].

The values of the Agile Manifesto promote a people-focused view of software development. It is no surprise, therefore, that researchers are now exploring various human and social aspects of Agile software development teams [39], [40], [4], [34], [6], [41], [42] in response to the Agile software movement's increasing popularity within industry over the past decade [43], [19], [34]. A systematic review of empirical studies of Agile software development found that about 20 percent of research studies on Agile software development focused on human and social factors [32]. The self-organizing aspect of Agile software development teams, however, has not been previously studied to any great extent.

Nerur et al. threw light on various issues related to transitioning into an Agile environment, broadly dividing them into technological, people-related, and process-related issues [34]. One of the people-related challenges is programmers used to solitary working styles moving into a collaborative environment. Collaborative decision making is predicted to be a challenge, requiring huge effort, time, and patience at the organizational level. Nerur's study further suggests that the traditional project manager's role of controller and planner would need to change to that of facilitator and collaborator. They also predict that the greatest challenge posed in the way of achieving this change would be for the managers to relinquish their authority [34].

A popular slogan "*people trump process*" highlights the importance of people in Agile software development [3]. Cockburn and Highsmith point out that while the success of any process is largely dependent on the people, the ability of the people to achieve their goals is dependent on the level of support they receive from users, customers, and management [3]. They argue that Agile organizations practice "*leadership-collaboration*" instead of command and control style management, and that management in Agile organizations trusts their teams to deliver to their best potential. They suggest that Agile teams function best in an organizational culture that supports people and collaborations.

Sharp and Robinson have conducted an extensive ethnographic study of five mature XP teams, describing characteristics of XP teams [6], collaboration and coordination in XP teams [44], the effect of different organizational cultures on the practice of XP [45], and the social aspects of XP's technical practices [46]. Their study confirms the highly collaborative and self-organizing nature of Agile teams [44]. Sharp and Robinson describe the culture of mature XP teams as possessing five characteristics:

1. respect on both an individual and team level,
2. responsibility on both an individual and team level,
3. maintaining quality of working life,
4. confidence in their own abilities coupled with constant revalidation and reaffirmation, and
5. trust, that underpins the other four.

Their study established the importance of story cards and story walls in collaboration and coordination within XP teams [6]. While simple, these physical artifacts proved to be information rich focal points for collaboration and coordination.

Williams et al. have extensively researched XP's pair programming practice [42], [47]. Pair programming has been shown to improve productivity and quality of products [47]. Transitioning from working alone into pair programming, however, can be challenging for programmers. Several practical tips are offered for programmers to enable a smooth transition to pair programming, including sharing all programming artifacts, such as design, code, etc.; taking turns to code and to review; remaining focused on the tasks; and receiving feedback to improve personal skills instead of being defensive and egotistic. Their study acknowledges that pair programming can be intense and mentally exhausting as it demands persistent focus. Pairs often take time off pair programming to attend to individual work.

In her doctoral research, Martin discovered that the customer role was generally played by a team of people, instead of by a single person as initially assumed in the literature [48], [49]. Martin's study describes an informal XP customer team that consists of different roles, where the Negotiator was the closest to the on-site customer defined in the literature. Martin's study also describes customer practices such as Customer Boot Camp and Pair Customering. These practices—when combined with the customer roles Martin identified—were found to help reduce the burden placed on the on-site customer role.

The social nature of Agile teams was explored through a Grounded Theory research study by Whitworth and Biddle [41]. The findings of their study highlight the importance of social and interaction-focused practices such as daily meetings, and the use of information radiators in establishing social answerability and awareness. The results emphasize the importance of self-organizing abilities of Agile teams, while highlighting the lack of research on the topic. Their study calls for more studies to be conducted on social and cultural issues on Agile teams, especially with regard to "*self-regulatory*" work structures [41].

Most of the above research has focused almost exclusively on XP teams [49], [50], [6], [44], [42]. In contrast, research on Scrum is scarce, despite Scrum being arguably the most popular Agile method used in the industry [32], [51].

2.4 Self-Organizing Teams

The concept of self-organizing teams was recognized long before it was incorporated as a hallmark of Agile software development [1]. This section presents a review of self-organizing teams from several perspectives: socio-technical systems, organizational theory, complex-adaptive systems, knowledge management, and finally, from the perspective of Agile software development.

2.4.1 Socio-Technical Systems Perspective

From a socio-technical systems perspective, research on self-organizing teams dates back to the Tavistock group's study of English coal miners as autonomous groups in the 1950s [52]. Autonomous groups were described as learning systems that expand their decision space in response to every day learning. The success of these autonomous groups was largely attributed to the supporting organizational environment, an informal structure with a decentralized, participative, and democratic system of control, called *concertive control* [53]. Concertive control was argued to be an alternative to the bureaucratic control marked by an hierarchical system with rational-legal rules rewarding compliance [54]. Self-managing teams were proposed as an exemplar of concertive control and were suggested to increase the organization's ability to respond to changing business conditions [53].

Self-managing teams were described as teams made up of 10 to 15 people taking on the responsibilities of their former supervisors whose everyday activities were guided by the senior management's corporate vision, who were cross-trained individuals setting their own work schedules, who displayed increased commitment to the company, and who coordinated with other areas of the company [53]. Self-managing teams in a concertive organization were said to be motivated by peer pressure as opposed to legal rules in a bureaucratic organization. The distinct synergy between the description of these self-managing teams and the theoretical concept of a self-organizing teams proposed in Agile software development is inescapable [7], [55].

2.4.2 Organizational Theory Perspective

Self-organizing teams have been described from an organizational theory perspective [56], [57], [58]. Morgan, in his book *Images of Organizations*, describes several metaphors for viewing an organization. One of the metaphors is *organizations as holographic brains*, which captures the concept of a hologram to represent organizations where the qualities of the whole system are captured in each of its parts. As a holographic brain, the organization or work group displays enhanced abilities to self-organize [59], [58]. Four principles of self-organization in a holographic organization are defined as: minimum critical specification, requisite variety, redundancy of functions, and learning to learn [60], [58].

Minimum Critical Specification refers to the senior management defining only the critical factors that are needed to direct the team and placing as few restrictions on the team as possible [58]. Morgan also emphasizes the need for self-organizing teams to work in an environment of "*bounded*" or "*responsible autonomy*" [58]. The role of management is extremely important in providing autonomy to the team and for team empowerment [56].

Requisite Variety and Redundancy of Functions: Morgan defines requisite variety as the need for any control system to match the complexity and diversity of the environment being controlled [58]. In other words, the organization must match the variability of its external environment. Requisite variety implies that changes in the environment of the organization are best handled by self-organizing teams. In other words, if the amount of variety or fluctuations in the environment is low, self-organizing teams—composed of members possessing a variety of skills—are not required. Self-organizing teams are effective when there are changes in the organizational environment. It is not surprising then that self-organizing teams are seen as improving the flexibility of an organization in terms of its ability to respond to change and as an influential factor in improving the quality of the employees' working life [56], [57].

The principles of requisite variety and redundancy of functions are closely related. Redundancy of functions refers to the multifunctionality of workers where workers are able to perform a wide variety of team tasks through cross training [56].

Learning to Learn refers to the team's ability to reanalyze problems, reappraise the best working method, and reconsider the required output if necessary [56]. Sustenance of self-organization requires double-loop learning, where the rules and norms adapt to changing environments [59].

The holographic organizations metaphor has been theoretically explored in the context of self-organizing Agile teams by Nerur et al. [35]. Minimum project planning and specification upfront on Agile projects is consistent with the principle of minimum critical specification. Interchangeable roles, multiple perspectives, and code ownership on Agile teams are theoretically consistent with the principle of requisite variety and redundancy of functions. The practices of refactoring, stand-up meetings, and pair programming are considered consistent with the principle of learning to learn (or double loop learning). Whether Agile teams are able to adhere to these principles in practice, however, has not been shown.

2.4.3 Knowledge Management Perspective

From a knowledge management perspective, one of the earliest papers to describe self-organizing teams, was "The New New Product Development Game" by Nonaka and Takeuchi, where they define a group as possessing self-organizing capability when it exhibits three conditions: autonomy, cross fertilization, and self-transcendence [28]. A team exhibits autonomy when they are provided freedom by their senior management to manage and assume responsibility for their own tasks and when there is minimum interference from senior management in the team's day-to-day activities [28]. A team exhibits cross fertilization when it is composed of individual members with varying specializations, thought processes, and behavior patterns and these individuals interact among themselves, leading to better understanding of each others perspectives [28]. A team possesses self-transcendence when they establish their own goals and keep on evaluating themselves so that they are able to devise newer and better ways of achieving those goals.

Self-organizing teams were seen as an important agent of knowledge creation and management in an organization [61]. Self-organizing teams accumulate and spread knowledge through 1) "*multilearning*," made up of multilevel learning across individual, group, and organizational levels, and "*multifunctional learning*" across functions, and 2) "*transfer of learning*" across different departments of the organization [28]. The self-organizing team with its cross-functional and multiple learning capabilities replaced traditional teams with specialists in particular knowledge areas.

2.4.4 Complex Adaptive Systems Perspective

Self-organization has also been discussed from the complex adaptive systems perspective [9], [62], [63], [64], [65], [28]. Complex adaptive systems are systems that exhibit spontaneous order through the process of self-organization [63]. Immune systems, ant colonies, human cities, and ecosystems are examples of complex adaptive systems [63]. Kauffman explored CAS in human organizations and economics, defining modern organizations as self-sustaining structure of roles and obligations [62]. Levin further suggested that cooperation and networks of interaction emerge out of individual behaviors and in turn influence them [64].

Anderson and McMillan [66] define self-organizing teams as teams that are

1. informal and temporary,
2. formed spontaneously around issues,
3. are not a part of a formal organization structure,
4. have a strong sense of shared purpose,
5. where team members decide their own affairs, and
6. where all members' primary roles relates to the task.

Augustine et al. compare Agile projects to Complex Adaptive Systems and suggest that the complex interactions among members leads to self-organization and emergent order [9]. Other proponents of this view insist that senior management and managers, while relinquishing control, must provide an environment that is conducive for self-organization to emerge [67], [65].

2.4.5 Agile Software Development Perspective

Finally, from an Agile software development perspective, self-organizing teams are considered the source of the best architecture, requirements, and design [1]. Self-organization is one of the principles behind the Agile Manifesto, having been identified as one of the critical success factors of Agile projects [9], [2], [1]. Self-organizing Agile teams are composed of "*individuals [that] manage their own workload, shift work among themselves based on need and best fit, and participate in team decision making*" [7]. Self-organizing teams must have common focus, mutual trust, respect, and the ability to organize repeatedly to meet new challenges [3].

Sutherland, a cocreator of Scrum, explains that self-organizing teams consist of "*members with diverse backgrounds*" who are "*given a free hand*" by the top management [68]. Schwaber, the other cocreator of Scrum, says that Agile methods "*employ self-organizing teams*" which are cross functional, not limited by their organizational job titles, training, or experience, rather the team "*self-organizes based on its strengths and weaknesses to do the work at hand*" [55]. Schwaber suggests individuals on the team need to coordinate their individual self-organization with the rest of the team via daily synchronization meetings called daily Scrums.

Larsen defines a self-organizing Agile team as a group of peers using one or more Agile methods that share a goal and accomplish the goal through collaboration [69]. The team approaches problem-solving collaboratively and strives for continuous improvement. Others have also mentioned the importance of self-organizing teams in Agile software development and the need for self-assignment, collective responsibility, cross functionality, and continuous learning in such teams [70], [71].

Self-organizing Agile teams are not leaderless, uncontrolled teams [3], [28]. Leadership in self-organizing teams is meant to be light-touch and adaptive [9], providing feedback and subtle direction [72], [28]. Leaders of Agile teams are responsible for setting direction, aligning people, obtaining resources, and motivating the teams [73].

In a longitudinal study of a single company adopting Scrum, Moe et al. studied barriers to self-organization by focusing on one aspect of self-organization—autonomy [51]. They found that management did not provide an environment conducive to self-organization that led to reduced external autonomy. They also report that high individual

autonomy proved to be a barrier to self-organization as members preferred individual goals over team goals.

Moe et al. also investigates the results of exploring the teamwork challenges that arise when introducing a self-managing Agile team [74]. The term *self-managing*, in that paper, is used to describe Agile teams and is considered synonymous with *autonomous* or *empowered* teams. The study uses Dickinson and McIntyre's teamwork model for understanding the self-managing nature of Agile teams, which includes components such as team orientation, team leadership, monitoring, feedback, backup, coordination, and communication [75]. The results show that the main challenges to achieving team effectiveness include problems with team orientation, leadership, and coordination, as well as highly specialized skills and corresponding division of work. The study suggests that trust and mental models, besides the components of Dickinson and McIntyre's teamwork model, are of great importance in understanding self-managing Agile teams. The study also recommends that both developers and management need to change in order to establish self-managing teams.

While the practitioner-based literature on self-organizing Agile teams abounds, the research literature on the subject is scarce. Some studies on Agile teams have acknowledged the self-organizing nature of Agile teams [6], [41]. Moe et al. note that Agile methods, specially Scrum, emphasize self-organizing teams but do not provide clear guidelines on how they should be implemented [74], [51]. Research on self-organizing Agile teams is limited to a single case-study-based research which explores one of the three conditions of self-organization—autonomy [51]. There is a lack of research exclusively focused on the self-organizing nature of Agile teams that extends across multiple organizations, countries, and cultures. This research has result in a grounded theory of self-organizing Agile teams that emerged from this research, in terms of their roles and practices, and the critical environmental factors that influence them.

3 Research Method

Grounded Theory is defined as "*a general methodology of analysis linked with data collection that uses a systematically applied set of methods to generate an inductive theory about a substantive area*" [76]. GT was developed by Glaser and Strauss [77].

The goal of GT is "*to generate a theory that accounts for a pattern of behaviour which is relevant and problematic for those involved*" [78]. In other words, GT tries to find the main concern of the participants and how they go about resolving it, through constant comparison of data at increasing levels of abstraction [76]. The nature of the "theory" generated by the Grounded Theory method is best understood as an explication of the research findings [79]. Grounded Theory has also been described as "*a general pattern of understanding*" [80]. In generating a theory, a GT researcher uncovers the main concern of the research participants and how they go about resolving it.

Differences between the two originators of Grounded Theory led to the emergence of two versions of the Grounded Theory method: Glaser's version of GT, often referred to as the Glasserian method or "classic" GT and Strauss' version, called Straussian GT [81], [82]. This research employs classic GT as it is the dominant form of GT used in software engineering research, and due to a larger number of resources available [83].

In the following sections, the main procedures of the GT method are described. Examples from the application of GT to this research are also included. A more detailed description of Grounded Theory and its application in our research is available elsewhere [14], [13].

This research started by exploring *Agile Project Management* as an area of research, primarily due to the growing popularity of Agile software development in software engineering research [84], [39], [49], [51], [34], [6], [41].

3.1 Minor Literature Review

Glaser strictly warns against extensive literature review in the *same area* of research during the *early stages* of the GT method [78]. A minor literature review was conducted in the beginning—just enough information on Agile methods was read to understand the basic facts and terminology in order to converse with the participants during interviews. A deeper understanding of Agile methods and, in particular, the self-organizing nature of Agile teams came mostly from the participants in the early stages of the research. Reading of substantive areas *different* from that of the research is considered vital in order for the researcher to understand how to apply the GT process [78]. Reading articles and dissertations describing research conducted using GT in other areas [85], [86], [87], [88], for example, was found to be useful.

3.2 Data Collection

After receiving Human Ethics Committee (HEC) approval, several Agile user mailing lists were contacted. The Agile Professionals Network [89] and the Agile Software Community of India [90] proved particularly useful as platforms to find research participants. While New Zealand was a convenient home ground, the Indian software industry was chosen because it is home to a well-established and flourishing software industry with an increasing number of Agile adoptions [91], [92], [93], [94], [33], [95].

The initial participants belonged to relatively new Agile teams and as such the emerging theory was mostly based around the initial challenges of becoming a self-organizing team. Using theoretical sampling, gaps in the emerging theory were discerned, which prompted the study more mature teams toward later stages of the research. A need to include participants from different areas of software development such as development, testing, management, etc., was also experienced at different stages of the research guided by the emerging theory. As a result, practitioners in a number of different organizational roles were approached, such as Agile coach (AC), developer, tester, business analyst (BA), designer, customer representative (Cust Rep), and senior management. Data collection by theoretical sampling helped develop the emerging theory by 1) adapting questions to focus on emerging concerns, 2) choosing participants that were well placed to provide information on the emerging concerns.

This research is based on 58 participants from 23 different software organization. Of these, 26 were from 10 New Zealand organizations, 28 were from nine Indian organizations, and four were from four organizations in North America. Interviews with Agile practitioners in New Zealand were conducted in Wellington. Interviews with

TABLE 1
Participants and Projects (P#: Participant Number, Position: Agile Coach, Agile Trainer (AT), Developer, Customer Rep, Business Analyst, Senior Management; *Organizational Size: XS < 50, S < 500, M $< 5{,}000$, L $< 50{,}000$, XL $> 100{,}000$ Employees)

P#	Positions	Method	Org. Size*	Location	Domain	Team Size	Project (months)	Iteration (weeks)
P1-P9	Dev x 3, BA, AC x 2, AT, Tester, Cust. Rep.	Scrum	M	NZ	Health	7	9	2
P10	AC	Scrum & XP	L	NZ	Social Services	4 to 10	3 to 12	2
P11-P18	Dev x 6, AC, SM	Scrum & XP	S	NZ	Environment	4 to 6	12	1
P19	SM	Scrum & XP	S	NZ	E-commerce	4	2	4
P20	AC	Scrum & XP	XL	NZ	Telecom & Transportation	6 to 15	12	4
P21	Cust. Rep.	Scrum	XS	NZ	Entertainment	6 to 8	9	4
P22	AC	Scrum & XP	S	NZ	Government Education	4 to 9	4	2
P23	AC	Scrum & XP	XS	NZ	Software Development	8	12	1
P24-P25	Dev x 2	Scrum	XS	NZ	Software Development	8 to 10	8	2
P26	AC	Scrum & XP	S	NZ	Farming	8	12	2
P27-P35	Dev x 4, AC, Tester, Sales Manager, SM x 2	Scrum & XP	S	India	Agile Software Development & Consultancy	5	6	2
P36-P39	AC x 4	Scrum & XP	M	India	Software Development	7 to 8	3 to 6	2
P40	SM	Scrum & XP	S	India	CRM and Finance	7 to 8	ongoing	3
P41	Designer	Scrum & XP	S	India	Web-based Services	5	1	2
P42	AC	Scrum & XP	L	India	Telecom	8 to 15	3	4
P43	AT	Scrum & XP	XS	India	Agile Training	7	8	2 to 4
P44-P45	Dev x 2	Scrum & XP	XS	India	Software Development	4	1	1
P46-P53	Dev, BA x 2, AT, AC, KS, HR, SM	Scrum & XP	M	India	Agile Software Products & Consultancy	15	12	1
P54	AC	Scrum & XP	M	India	Financial Services	8 to 11	36	2
P55	AC	RUP	XS	Canada	Telecom	10 to 15	10 to 15	2 to 4
P56	SM	Scrum	M	USA	Oil and Energy	5 to 8	12	2
P57	Cust. Rep.	Scrum & XP	M	USA	CRM and Cloud Computing	variable	variable	2 to 4
P58	AC	Scrum & XP	XS	USA	Health	variable	variable	2 to 4

Agile practitioners in India were conducted in New Delhi, Mumbai, and Bangaluru (previously called Bangalore). The remaining few interviews with North American participants were conducted during the Agile 2008 conference in Toronto. The domains included health, social services, telecommunications, entertainment, agriculture, oil and energy, etc. Some companies were not only practicing Agile as a software development method but also developing products to cater to Agile teams, such as Agile project management tools, and therefore their domain is also termed Agile software development.

The products and services offered by the participants' organizations included web-based applications, front and back-end functionality, and local and off-shored software development services. The project duration varied from two to 12 months and the team sizes varied from 2 to 20 people on different projects. The organizational sizes varied from 10 to 300,000 employees. Table 1 shows participant and project details.

Participants were practicing Scrum or a combination of Scrum and XP. All participants were practicing fundamental Agile practices such as iterative and incremental development (with varying iteration lengths), iteration planning, estimation and planning of user stories and tasks, testing, status report meetings (such as daily stand up), frequent release of working software, and some form of retrospective meetings. A majority of the participants engaged in test-driven development and pair programming (on demand). Some participants were certified Scrum Masters. Several participants were active in local and international Agile communities—speaking at events and authoring Agile-related articles online.

Participants varied in their experiences of working on Agile projects; while some were very fresh (first Agile project), some others had experienced working on a number Agile projects, and others had more than five years of experience on Agile projects.

3.2.1 Interviews and Observations

Data were collected through interviews and was supplemented by observations over a period of four years. Face-to-face, semistructured interviews with Agile practitioners were conducted using open-ended questions. The interviews were approximately an hour long and focused on the participants' experiences of working with Agile methods. In particular, the challenges faced in Agile projects and the strategies used to overcome them were discussed. While the interviews were largely conversation driven, some standard questions asked were:

- *Please can you tell me about your professional background?*
- *What is your role on the project?*
- *What are the major challenges you've faced on this project, because you were practicing Agile?*
- *How did you overcome that [challenge]?*

Data collection and analysis were iterative so that constant comparison of data helped guide future interviews and the analysis of interviews and observations fed back into the emerging results. As the data was analyzed and new concepts and categories emerged, the subsequent interview questions were updated to focus on the emerging codes. For example, questions in later interviews were modified to

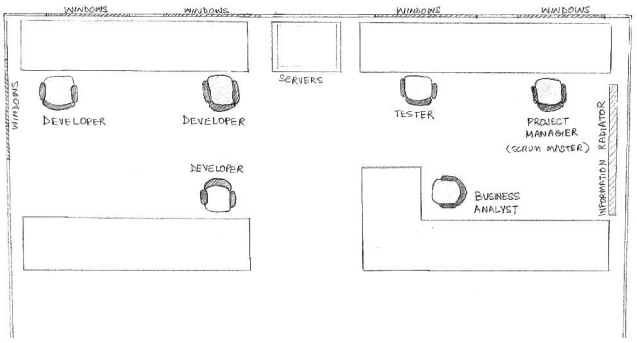

Fig. 1. Physical setup of an open-plan workspace.

focus on the main concern of the participants, i.e., becoming a self-organizing Agile team:

- *Do you believe that your team is self-organizing? If yes, why? what makes you self-organizing?*

In addition to interviews, observations were made about the participants' workplaces, such as seating and setup of information radiators, and several Agile practices, such as daily stand-up meetings (colocated and distributed), release planning, iteration planning, and demonstrations. Fig. 1 shows the physical setup of an NZ team.

Face-to-face interviews provide the opportunity not only to record verbal information but also the mannerisms, actions, and expressions which add to the verbal information. Conducting semistructured interviews, instead of completely structured interviews, helped uncover the real concerns of participants rather than forcing a topic on them.

The majority of the interviews were first voice recorded and then transcribed. A small number of interviewees were not comfortable being recorded, and so handwritten notes were taken. The interview transcripts served as a good starting point for analysis.

3.3 Data Analysis

Data analysis—called *coding* in GT—can begin as soon as some data have been collected. There are two types of codes produced as a result of data analysis or coding: substantive codes and theoretical codes. The substantive codes are "*the categories and properties of the theory which emerges from and conceptually images the substantive area being researched*" [96]. In contrast, theoretical codes "*implicitly conceptualize how the substantive codes will relate to each other as a modeled, interrelated, multivariate set of hypothesis in accounting for resolving the main concern*" [96]. The following sections describe the coding mechanisms—*open coding* and *selective coding*—that lead to substantive codes and *theoretical coding* that leads to theoretical codes.

3.3.1 Open Coding

Open coding is the first step of data analysis. Open coding was used to analyze the collected data in detail [78], [97]. To explain open coding, an example of working from interview transcripts to results for the category "*Mentor*" is presented, which is one of the self-organizing Agile team roles.

Open coding begins by collating key points from each interview transcript. Then, a code—a phrase that summaries the key point in two or three words—is assigned to each key point [98]:

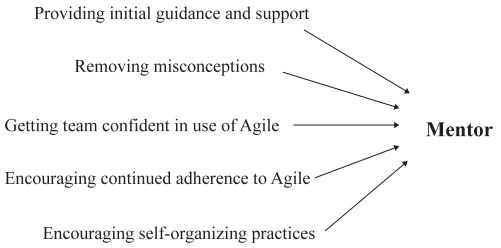

Fig. 2. Example of emergence of a category from underlying concepts.

Interview quotation: "*We had [Mentor] as well at the time [the team started Agile practices] so...It made it easy...having [Mentor] there as a backup... [it has] been really good to have that guidance from [the Mentor].*"—P8, Tester, New Zealand

Key Point: "*Coach providing guidance in initial stages*"

Code: Providing initial guidance (P8, NZ)

Line by line data analysis is more effective and useful than word-by-word analysis, which can be tedious and potentially misguiding [99]. The use of key points made it easy to focus while coding [99].

3.3.2 Constant Comparison Method

GT's *constant comparison method* involves constantly comparing codes arising out of each interview against other codes from the same interview and those from other interviews and observations [77], [76]. The constant comparison method was used again to group these codes to produce units of a higher level of abstraction, called concepts in GT.

Concept: Providing initial guidance and support

Removing misconceptions, encouraging self-organizing practices, getting the team confident in the use of Agile methods, and encouraging continued adherence to Agile were the other codes that emerged. Finally, the *constant comparison method* was repeated on concepts to produce a third level of abstraction called *categories*.

Category: Mentor

A *Mentor* is one particular individual in the Agile team who assumes the responsibility of providing guidance on the chosen Agile method. Fig. 2 shows the emergence of the category *Mentor* from underlying concepts. Examples of using diagrams to represent emergence of concepts from data analysis in GT studies are based on [99], [98].

Fig. 3 depicts the levels of data abstraction in GT. Other codes, concepts, and categories emerged in a similar manner. Emergence of the different categories is presented in similar diagrams throughout this paper.

The rigor of the GT method is embodied by the constant comparison method. This process is repeated every time a new category is found or there are changes in an existing category or new properties of an existing category is discovered, leading the researcher to revisit previously coded transcripts to see if they have the new property.

The observations were analyzed and compared to the concepts derived from the interviews. The observations did not contradict (but rather supported) the data provided in interviews, thereby strengthening the interview data.

3.3.3 Core Category

The emergence of a *core category* [76] marks the end of open coding. The core category "*accounts for a large portion of the*

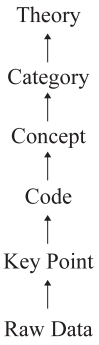

Fig. 3. Levels of data abstraction in Grounded Theory

variation in a pattern of behaviour" and is considered the "*main concern or problem*" for the participants [78]. The core category captures the main concern of the participants, which becomes the research problem.

Some criteria for choosing the core category include: It must be central, it must be related to several other categories and their properties, it must reoccur frequently in the data, it must relate meaningfully and easily with other categories, and it must account for most variations in data [78]. The category that passed all the criteria for core was *self-organizing Agile teams*.

3.3.4 Selective Coding

With the emergence of a core category, the researcher ceases open coding and moves into *selective coding*. Selective coding involves selectively coding for the core category by limiting the coding to "*only those variables [concepts or categories] that relate to the core variable [category] in sufficiently significant ways as to produce a parsimonious theory*" [78], [82]. The core category guides further data collection, analysis, and theoretical sampling [78].

When further data collection and analysis on a particular category leads to a point of diminishing results, the category is said to have reached *theoretical saturation* [76]. The researcher can stop collecting data and coding for that category. In this research, the last few interviews provided no new insight into the existing categories, which was clear indication of theoretical saturation.

3.4 Memoing

Memoing is the ongoing process of writing memos throughout the GT process. Memos tend to be free-flowing ideas about the codes and their relationships. Memos were written down as ideas about the emerging codes and their relationships occurred. Memoing is a simple but powerful way to allow all the ideas and thoughts about a certain code, concept, or category to pour out. With further data collection and analysis, memos were modified to reflect new ideas. Memoing allowed the relationship between different concepts and, later, between different categories, to emerge as the similarities or differences between each or how one affected the other were noted down.

3.5 Sorting

Researchers can begin sorting the theoretical memos once data collection is nearly finished and coding is almost saturated. Sorting the memos forms a theoretical outline [78]. The advantage of sorting is that it "*puts the fractured data back together*" [78]. Care was taken to sort ideas, not data. Chronological ordering is not the purpose of sorting; instead, sorting is done on a conceptual level, resulting in an outline of the theory in terms of how the different categories relate to the core-category.

3.6 Major Literature Review

The literature on self-organizing Agile teams was reviewed once the findings seemed sufficiently grounded and developed. The purpose of a major literature review *after* analysis is to 1) protect the findings from preconceived notions and 2) to relate the research findings to the literature through integration of ideas [78].

3.7 Theoretical Coding

Theoretical coding is defined as "*the property of coding and constant comparative analysis that yields the conceptual relationship between categories and their properties as they emerge.*" [76]. Theoretical coding involves conceptualizing how the categories (and their properties) relate to each other and how they can be integrated into a theory [78].

This research resulted in a grounded theory of self-organizing Agile teams which presents the three main themes that emerged from the study, namely, self-organizing roles, self-organizing practices, and critical factors influencing self-organizing teams. In particular, the grounded theory of self-organizing Agile teams explains how software development teams take on informal, implicit, transient, and spontaneous roles and perform balanced practices while facing critical environmental factors. These roles include Mentor, Coordinator, Translator, Champion, Promoter, and Terminator [15]. The practices involve balancing freedom and responsibility, cross functionality and specialization, and continuous learning and iteration pressure [100]. The factors are senior management support [11] and level of customer involvement [10], [12]. The categories self-organizing practices and critical environmental factors have been described elsewhere [100], [10], [11], [12], [13].

A preliminary description of the category "self-organizing team roles" was presented in our conference paper [15]. Grounded Theory is based on analysis of participants self-reports (and observation) and they reported about these particular roles (Mentor, Coordinator, Terminator, Champion, Promoter, and Terminator) as roles that enable self-organization, i.e., self-organizing roles. At the scale of our study, no other roles emerged anywhere near as strongly as these roles, based on Grounded Theory's constant comparison method. In the following sections, we describe the self-organizing roles on Agile software development teams in detail as they evolved through and finally appeared out of the four year study. We have selected quotations drawn from our interviews that shed particular light on the concepts.

4 Results: Self-Organizing Team Roles

This section describes the key results of the study—the self-organizing roles that exist on Agile software development teams. Members of software development teams, both Agile and non-Agile, fulfill organizational roles on the team. For example, developers are responsible for development, testers are responsible for testing, business analysts are

TABLE 2
Self-Organizing Agile Team Roles

Role	Definition	Interacts with	New Teams	Mature Teams
Mentor	Guides and supports the team initially, helps them become confident in their use of Agile methods, ensures continued adherence to Agile methods, and encourages the development of self-organizing practices in the team.	Team, (Senior Management)	AC	Anyone
Co-ordinator	Acts as a representative of the team to manage customer expectations and co-ordinate customer collaboration with the team.	Team, Customers	Dev/ BA/AC	Anyone
Translator	Understands and translates between the business language used by customers and the technical terminology used by the team to improve communication between the two.	Team, Customers	BA	Anyone
Champion	Champions the Agile cause with the senior management within their organization in order to gain support for the self-organizing Agile team.	Senior Management	AC	Anyone
Promoter	Promotes Agile with customers and attempts to secure their involvement and collaboration to support the efficient functioning of the self-organizing Agile team.	Customers	AC	Anyone
Terminator	Identifies team members threatening the proper functioning and productivity of the self-organizing Agile team and engages senior management support in removing such members from the team.	Team, (Senior Management)	AC	Agile Coach (+ whole team)

Agile coach, developers, business analyst.

responsible for requirements analysis, etc. In Agile teams, however, these organizational roles are not strictly adhered to, and members often function outside their boundaries when organizing themselves. Members of Agile teams play one or more of six informal, implicit, transient, and spontaneous roles in order to self-organize. These self-organizing Agile team roles—*Mentor*, *Coordinator*, *Translator*, *Champion*, *Promoter*, and *Terminator*—are focused specifically toward self-organization and are performed over and above the organizational roles. The self-organizing roles are informal and implicit because, unlike organizational roles, they are not formally designated to the individuals who play them. The self-organizing roles are transient because, unlike organizational roles, they emerge in response to challenges faced by the Agile team and disappear or become dormant as the problems subside. The self-organizing team roles are spontaneous because, unlike organizational roles, they are intuitively picked up by different members of the team. Table 2 provides an overview of self-organizational Agile team roles.

The following sections describe each of these self-organizing roles in detail. The descriptions include selected quotations drawn from the interviews that shed particular light on these categories and that are spread across participants, geographically and by their organizational roles.

4.1 Mentor

Guides and supports the team initially, helps them become confident in their use of Agile methods, ensures continued adherence to Agile methods, and encourages the development of self-organizing practices in the team.

The initial stages of becoming a self-organizing Agile team can be very difficult. Many participants described the transitioning phase as "*difficult*," "*a challenge*," "*a struggle*," and "*a war*" (P15, P25, P36, P56). During the initial stages of transitioning, the team's existing work environment and practices must be changed to become Agile. At this stage, a *Mentor*, typically played by an Agile Coach (Scrum Masters and XP Coaches), teaches the new team about Agile software development [15]. A description of how this category emerged from data analysis has been provided in Section 3.3. Fig. 4 illustrates how the category *Mentor* emerged from the underlying concepts.

4.1.1 Providing Initial Guidance and Support

The *Mentor* familiarizes the team with the Agile Manifesto [1] values and principles, and informs them of one or more particular Agile methods, such as Scrum and XP. The theoretical knowledge of Agile software development and the practices of particular Agile methods are imparted by the *Mentor* in several ways. Some *Mentor*s have informal talks with their teams, while others conduct more formal training sessions spanning a few days.

Most team members perceive the Agile practices to be simple enough to comprehend, but when it comes to implementing them on a daily basis, they need guidance and support. The *Mentor* oversees the new team as they begin to practice Agile software development on a day to day basis.

"It's more important that you get everything right at the start. Because the process itself is not that complicated [but] doing things

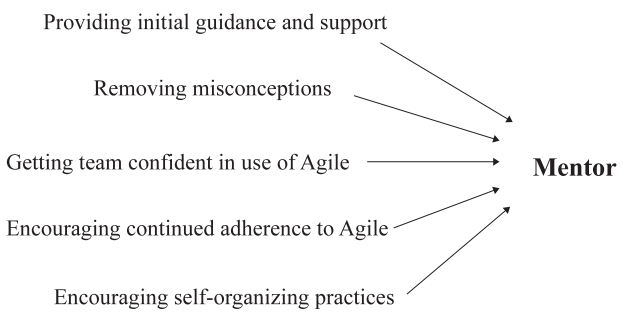

Fig. 4. Emergence of the category *Mentor* from underlying concepts.

along the lines of the process is a little bit harder than the process itself...So with [the Mentor] it was kind of to teach us how Agile works and shape our mindset and make sure everyone knows how to work under the Agile umbrella."—P1, Developer, New Zealand

As the team members learn and practice Agile software development, they are faced with several challenges. Finding their place and role in the new team is one of these challenges. Team members often perceive the changes as a criticism of their personal skills and retreat into a defensive corner, shunning the changes brought on by the introduction of Agile methods. A *Mentor* is quick to identify these insecurities among team members and proactively tries to clear the air of negativity from the team by encouraging them to focus on the reevaluation of their work environment instead of their own personal skills:

"*All the dirty doings get exposed. Hand holding people at that time...trying to take away the finger pointing...People go into defensive mode...that's when whole negativity comes in and all Agile practices are thrown out to the wind!...[encourage] focusing on what essential good practices, fundamental framework which has to be put in place.*"—P36, Agile coach, India

Sometimes, a *Mentor* steps in to **remove misconceptions** about Agile among team members. As one of the *Mentors* disclosed:

"*We were establishing from the start and...It's mainly been showing people through that process...It's a matter of overcoming and explaining the misconceptions.*"—P10, Agile Coach, NZ

The *Mentor* encourages the team members to voice their opinions and concerns freely, thereby creating an environment of trust in the team. Once the team members vocalize their concerns, the *Mentor* helps them overcome their problems.

4.1.2 Getting the Team Confident

As the team moves through sprints or iterations, they become more confident in their understanding and practice of Agile methods. Demonstrations of working software to the customers and receiving feedback from them at the end of the sprint become important sources of positive reinforcement for the new team. The *Mentor* encourages the team to take the feedback in a constructive spirit and use it to improve their practices.

"*When you get the team used to success, that's where a change happens in them. You'll have a team that starts...they haven't done this before, they don't quite know how to do it. You need to show them...that they have achieved something, that they had a client presentation and the software worked...And with the next iteration...they get a little bit more confidence...And after a few such validation cycles, then they start to get confident.*"—P20, Agile Coach, NZ

4.1.3 Encouraging Continued Adherence

Inexperienced or fresh members of the team, with no previous software development experience, find it easier to adopt Agile practices.

"*I find that there are perfectly capable developers that for one reason or another are not bothered to change anymore. They [experienced developers] have achieved a certain level of perceived mastery and they're not at all driven to excel or to challenge themselves...And conversely, you have hungry people [fresh developers] that don't know any better just yet and you can show them a way to do better, and they do.*"—P20, Agile Coach, New Zealand

The more mature team members, however, with previous experience of working with non-Agile software development methods, have a tendency to revert to their old ways in the initial stages.

"*Actually it takes a lot of effort for a team to become self-organizing, specially if people are coming from traditional software development methods. It takes time, specially because I've worked with [company name] and even in [this company] you see people they come from traditional, they are into a habit of work which is very hard to leave to start with.*"—P31, Agile Coach, India

An important aspect of the *Mentor* role is to highlight the importance of continued adherence to Agile principles and values. The following quote describes a project where the *Mentor* was prematurely let go after the management perceived the team to be self-organizing and no longer in need of support. This turned out to be a huge mistake. In the absence of a *Mentor*, the team lost the importance of retrospectives.

"*In the [retrospective] that we do they are so much quicker now than it used to be when we had [the Mentor] with us...[the Mentor] didn't have a vested interest in the product, she had a vested interest in the team...And now it is almost like lip service...we don't do self-evaluation as well as we used to.*"—P8, Tester, New Zealand

In relatively new teams (usually less than a year of experience), the role of the *Mentor* is taken up by experienced Agile coaches, who display a firm understanding of both Agile methods and their teams' issues. These Agile coaches are often employed on a contractual basis to guide the new team during the initial stages of practicing Agile software development. In more mature Agile teams (fluent in the use of Agile practices, for usually more than a year), however, the role of the *Mentor* is taken up by anyone in the team with wide experience in Agile software development. For example, in one of the Indian Agile organizations, most members have several years of experience in Agile software development and do not need a full-time *Mentor*. Whenever a newcomer joins the team, one of the senior members takes up the role of the *Mentor* and helps them become accustomed to the teams' Agile practices. A similar trend was noticeable in New Zealand teams.

"*I've been mentoring [a new team initially]...[now] the more senior of the two BA's [business analysts] is taking a [Mentor] role.*"—P26, Agile Coach, NZ

In a mature Agile team, senior members are expected to be able to mentor newcomers on a team:

"*you're a very senior [developer] about 8 to 10 years and you are going to pair up with a junior, to be able to match up to his expectations and improve him or mentor him, based on your knowledge.*"—P52, Human Resource Manager, India

The mentor role emerges on a need-basis, displaying the transient and spontaneous nature of this self-organizing role.

4.1.4 Encouraging Self-Organizing Practices

Over time, the *Mentor* helps team members learn and perform Agile practices that achieve and sustain self-organization. These practices include collective estimation and planning, self-assignment, self-evaluation through retrospectives, etc. A few examples of these practices and how the *Mentor* encourages them are provided here.

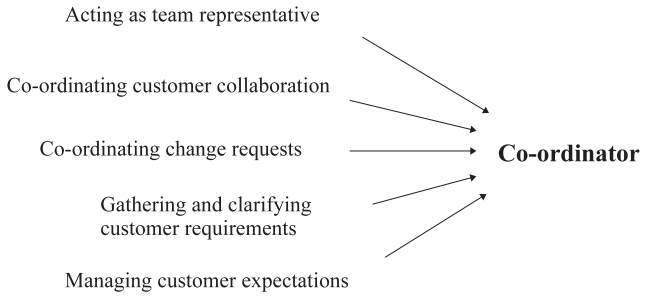

Fig. 5. Emergence of the category *Coordinator* from underlying concepts.

The *Mentor* helps team members practice estimation and planning. Project planning and estimation in traditional software development projects is mostly done by the project managers and does not involve team members. As such, many team members in a new Agile team with previous experience of working in traditional software development environments have never been involved in project planning and estimation. Therefore, the importance of a *Mentor* in guiding team members through estimating and planning for Agile projects is immense.

Similarly, the *Mentor* helps the team learn and practice self-assignment.

> "*It took them [new Agile team] a bit of time to stop coming and asking us what they should be working on and the answer was always "pick one!" And after while it became natural...people were picking stuff...and that worked really well.*"—P25, Developer, New Zealand

4.2 Coordinator

Acts as a representative of the team to manage customer expectations and coordinate customer collaboration with the team.

Agile methods expand the customer role within the entire development process by involving them in writing user stories, discussing product features, prioritizing the feature lists, and providing rapid feedback to the development team on a regular basis [101], [102], [49], [34]. These collaborative activities are difficult to coordinate with the customer for various reasons, such as physical distance between the development team and their customers, lack of time commitment on the part of the customers, and ineffective customer representation [12], [10]. The *Coordinator* role emerged on Agile teams to overcome these challenges and facilitate collaboration with customers [15]. Fig. 5 illustrates how the category *Coordinator* emerged from the underlying concepts.

4.2.1 Coordinating Customer Collaboration

In the context of the Indian software industry, Agile teams often face off-shored customers. Coordinating with customers across geographic and time zone differences is a challenge for Indian teams. The teams find it useful to have someone **acting as a team representative** coordinating between the team and their distant customers representatives. In one of the Indian projects, the *Coordinator* role was played by a developer who helped coordinate with off-shored customers:

> "*Initially we avoided [having team leads]...but sometimes, because we are working offshore [it is] good to have one person who can communicate. Not a team lead in the sense not telling people what to do [but] more like co-ordinator—talks to everybody.*"—P34, Senior Management, India

The *Coordinator* interacts with the team on a regular and intimate level and coordinates communication between the team and the customers:

> "*We assign a customer representative who interacts with the team ... but then passes on the feedback from the customer to the team and vice versa.*"—P54, Agile Coach, India

Initial analysis of new Agile teams in New Zealand revealed that teams face similar problems with distant customers and make use of a *Coordinator* to facilitate customer collaboration. In the case of a New Zealand team, a business analyst on the team acted as the *Coordinator*, representing the team to their customers and coordinating communication efforts.

> "*...it makes sense to have a [Co-ordinator] in the middle...if you have some sort of problem, you don't have five people asking the same question at the other end; which normally business people don't like...so having [the business analyst] as a [Co-ordinator], it's working for us.*"—P1, Developer, New Zealand

Some New Zealand practitioners found their respective customer representatives to be ineffective in providing timely requirements and feedback, while others found them lacking in proper understanding of Agile practices.

> "*Unfortunately the person who is [the customer rep] has an I.Q. of literally 25...doesn't really know how the current system works, doesn't know much about the business process, is petrified of the project sponsor, and is basically budget-driven. So she doesn't really care if it's not going to work in a way that the end users like.*" (undisclosed) Developer

In contrast, an effective customer representative was described as "*someone who understands the implications of that system...where it fits into the business process*" and at the very least "*someone who knows how to use a computer!*" (P10, P8).

A *Coordinator* is useful in such situations where the customer representative is unable or unwilling to devote the amount of time that the teams require to collaborate [10], [12]. Similarly, the *Coordinator* role helped facilitate collaboration with customer representatives that the teams perceived to be largely ineffective.

4.2.2 Coordinating Change Requests

The *Coordinator* also helps coordinate change requests made by the customers. Responding to change [1], [19] is an integral part of Agile methods and a *Coordinator* helps in dealing with changes in a systematic way so that the team can respond to them effectively:

> "*[the Co-ordinator] still needs to get all the requirements to us, so whenever the business owner wants to make a change...we can plan a little bit ahead; [The Co-ordinator] might say 'OK guys, this might come in the next couple of sprints, think about it and figure out how to handle it'. So that's kind of cool.*"—P1, Developer, New Zealand

The team needs a clear list of requirements (Scrum's product backlog) prioritized by the customer before they can begin their development iteration. The *Coordinator* is

responsible for **gathering and clarifying customer requirements** and priorities.

> "*If [the Co-ordinator] is not there things sort of stop spinning. A lot of the time we have to come back to him: 'Is this important? Is this prioritized?...when the client says 'Oh, that's all priority' we have to go back and say 'Which?! What do you mean?!' So then [the Co-ordinator] has to go back and say 'you can't have all priority!'*"—P2, Developer, New Zealand

In another New Zealand team with a distant customer (in a different city), a couple of developers had taken on the role of *Coordinators*, coordinating change requests.

Observation of a Team Meeting, New Zealand

> "*The Agile coach asked everyone to gather around the table at the center of the room. This was a combined meeting for all the three teams to discuss some interdependencies and clarify requirements. One of the team members who had been in direct contact with the customer played the role of [the Co-ordinator] on the meeting, providing requirements and clarifying doubts for the team (based on the information provided by the real customer). It was obvious that the Co-ordinator was in regular contact with the customer as he talked to the team pretending to be real customer. The team laughed at certain jokes about the requirements and how it was natural for the real customer to always request certain features. The Co-ordinator made the team aware of the customer requirements. As the Agile coach later confirmed, the customer had provided 3 individuals to be in contact with the Co-ordinators on the team regarding the project. The Agile coach was satisfied with the level of customer involvement.*"

When asked about these *Coordinators*, other team members confirmed that the two developers had taken up the responsibility of collaborating with the customers spontaneously in response to the problem of the entire team coordinating across distances. These two developers were better communicators compared to the rest of the team and had spontaneously taken on the *Coordinator* role.

> "*We've got two people that have...I'm just trying to think...no one ever said 'you guys, that's your role' but it's just developed that way. And probably more so from their ability to communicate ideas; they're well-spoken and able to get those ideas across...Which is great for developers!*"—P13, Developer, New Zealand

4.2.3 Managing Customer Expectations

Another part of the *Coordinator* role is to manage customer expectations. It takes time for a new Agile team to become fluent in Agile methods and reach a state of stability and performance. In the meanwhile, the first few sprints are challenging for the team and they experience high fluctuations in team velocity. During this crucial initial stage, the *Coordinator* carefully manages customer expectations:

> "*I have sort of a secret conversation with the customer, 'right okay, this team is new here for learning, expect them to blow the first sprint, it is very likely to happen'...and if anything good comes out of it, they [customers] are positively surprised.*"—P23, Agile Coach, NZ

On a relatively new Agile Indian team, the *Coordinator* role was played by a developer who interfaced with off-shored customers on behalf of the team. On a relatively new New Zealand team, the *Coordinator* was played by a business analyst facing the customers as a team representative. As the research progressed and more mature Agile teams were included, we found that the role of the *Coordinator* could be taken up by anyone in the team, not necessarily the business analysts or developers. Most members of mature self-organizing Agile teams are capable of playing the *Coordinator* role and coordinate with customer representatives directly.

> "*Sometimes we have the voice chat [with the customer representative] and these days we have the text chat. It lasts around half an hour on the minimum side and on the maximum side 3 hours or 4 hours.*"—P44, Developer, India

> "*Everyone does that [talk to the customer]. We are all on Skype. We added ourselves to a group...and then we just chat, even if I talk to the customer, the other person [team member] also knows what I'm talking because maybe tomorrow they face the same question so they can just observe the conversation.*"—P29, Developer, India

In both new and mature teams, the *Coordinator* role exists despite the presence of the *Mentor*.

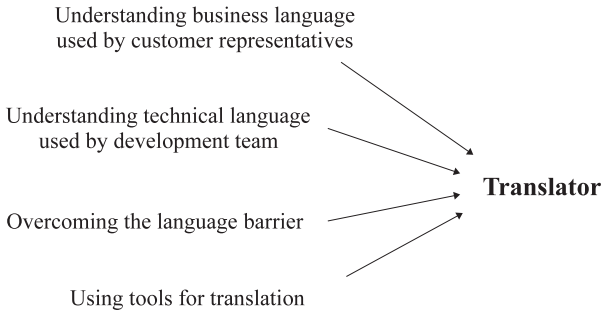

Fig. 6. Emergence of the category *Translator* from underlying concepts.

4.3 Translator

Understands and translates between the business language used by customers and the technical terminology used by the team, to improve communication between the two.

Development teams and their customer representatives use different languages when collaborating on Agile projects [10], [103], [12]. While the development teams use a more *technical* language composed of technical terminology, their customers use a more *business* language composed of terminology from the customers' business domains. There is a need for translation between the two languages in order to ensure proper communication of product requirements from the customer representatives and clarification of issues from the development team side. The *Translator* role emerged in self-organizing Agile teams to overcome the language barrier [10], [103], [12]. Fig. 6 illustrates how the category *Translator* emerged from the underlying concepts.

4.3.1 Overcoming the Language Barrier

Self-organizing Agile teams are responsible for collaborating effectively and frequently with customer representatives to elicit product requirements. In Scrum and XP, user stories are written down on story cards by customer representatives in the *business*'s language with domain specific requirements. The development team needs technical tasks written in *technical* language that are specific enough for development to commence. The actual translation of business

requirements into technical tasks happens when user stories are broken down into technical tasks:

> "*The biggest issues with the development team...the translation of what the client wants into something the development can create. So you have a story card with some features on....how to turn that story card into part of a website?*"—P19, Senior Management, New Zealand

The language barrier between development teams and their customers poses a threat to effective team-customer collaboration by limiting their understanding of each other's perspectives. The *technical* language used by development teams was difficult for their customers to understand:

> "*(Laughs) The client always expects that the information they sent to the development team will be enough... We have meetings with them and obviously there are some gaps in the language and in the jargon... I think... technical language is a problem for business people obviously.*"—P14, Developer, New Zealand

> "*I might explain something in a very cryptic, technological way and [the customers] won't understand a word!*"—P2, Developer, New Zealand

Business people, such as customer representatives, "*switch off*" when they are "*provided information with a technical bent*" (P22). Similarly, the customers' *business* language was difficult for the development teams to understand, as a Scrum Product Owner (customer representative) noted:

> "*They are very smart developers and they are really into 'yes we can code this or make this thing', but not really putting themselves in the user's shoes or the client's shoes.*"—P21, Product Owner (customer representative), New Zealand

Initial data analysis revealed that the role of the *Translator* was most often played by business analysts (P1, P2, P4, P8-P10). Business analysts were considered suitable candidates for the *Translator* role because of their ability to **understand both *technical* and *business* languages** and to act as a bridge between the two. The need for a "*good BA*" was evident on some teams (P4, P15, P21, P23) suffering from the language barrier. As the research progressed, more mature teams were studied and it became apparent that the *Translator* role was not limited to professional analysts, and could be played by anyone on the team with good communication skills and understanding of business concerns.

> "*...translators...understand the concerns of the business and translate them into priority elements that the development group can actually focus on to achieve...Somebody who wants to do it, who has this compulsion 'let me translate, let me help'...sometimes a PM, sometimes it's a BA, sometimes it's a developer, a tester.*"—P20, Senior Agile Coach, NZ

In relatively new teams, the *Translator* role was taken up spontaneously by team members. In more mature teams, participants ensured that they were "*hiring smart, pragmatic communicators*" upfront (P10, P52) with innate *Translator* skills when recruiting for an Agile team.

> "*strong public-oriented skills...to solve the business problem of the customer...more important to understand the customer and their requirements...you have to be very smart enough to get the requirements [and] understand the business intent when you solve a problem.*"—P52, Human Resource Manager, India

4.3.2 Using Translator Tools

There are several tools that help team members take on the *Translator* [103]. These include: a project dictionary, using iterative reasoning, and encouraging cross functionality in the team.

One of the Indian teams uses a "*project dictionary*" to assist everyone on the team in becoming a *Translator*. This dictionary is an online editable document (wiki) populated by the customers with business terms, their meaning, and their contexts of use. These business terms are translated directly into code by the team using the same variable names, providing a mapping between the customers' business terms and their technical implementation for a given project. The customer representatives are able to view and edit the contents of the evolving dictionary.

> "*we have extensive documentation...a wiki [where the customers] have explained their whole infrastructure...as and when they build up the requirements they come and edit the document...its kind of like a glossary and also the rules that figure in that world of theirs...we capture all that and ensure our domain is represented exactly like that in code.. ..so when they say 'a port has to be in a cabinet which has to sit in a rack' it directly translates to code!*"—P46, Developer, India

Another *Translator* tool is iterative reasoning—questioning proposed technical solutions repeatedly until the abstract business reasoning behind the technical details is evident.

> "*why do we need that database back up procedure? or...database recovery? and it's right down at the technical level [asking] the question why, why, why till...you'll eventually discover there's a good business reason for having it.*"—P22, Senior Management, New Zealand

Using iterative reasoning, technical solutions could be abstracted to higher levels till they were clearly aligned with their business drivers.

Interactions between members from diverse disciplines fosters understanding of the project from multiple perspectives [28]. As the team learns to understand their customer's perspective, they achieve greater levels of cross functionality and are able to translate between their respective languages. An experienced Agile coach disclosed that the secret to acquiring the *Translator* skills is through cross functionality.

> "*The whole thing with Agile is getting people to be more cross-disciplinary, to take an interest in somebody else's perspective...The moment you understand that cross-concern, you're teaching everybody to become a translator.*"—P20, Agile Coach, NZ

Relatively new Agile teams often have one or two individuals playing the *Translator* role based on either their personal abilities or professional skills. In contrast, most members of mature Agile teams are bilingual—speaking *technical* language in development circles and translating *business* language when collaborating with customers. The skills of a *Translator* can be an attribute of professional training (business analysts), natural abilities (natural communicators), or can be acquired using existing Agile practices such as cross functionality and adapted practices such as a dictionary and iterative reasoning.

Both the *Translator* and *Coordinator* roles interact with the team on one side and the customers on the other. The *Translator* role is distinct from the *Coordinator* role in that the *Coordinator* role emerged in response to problems around collaborating with distant, unavailable, or ineffective customers. The *Translator*'s role, on the other hand, involves translating ideas in expressions that the business/customer representatives understand into terminology that

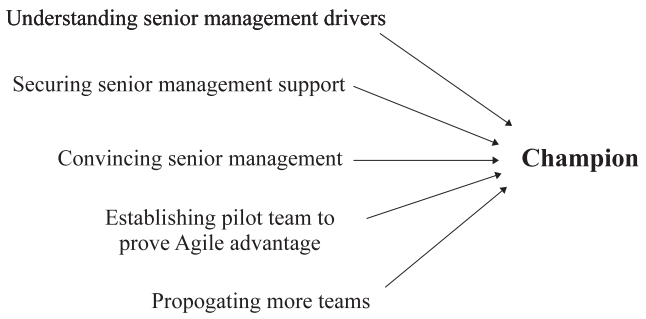

Fig. 7. Emergence of the category *Champion* from underlying concepts.

the development team is familiar with and vice versa. They were, in some cases, played by the same person.

4.4 Champion

Champions the Agile cause with the senior management within their own organization in order to gain support for the self-organizing Agile team.

Self-organizing Agile team cannot emerge and flourish in isolation [15]. The importance of senior management support in establishing and propagating self-organizing Agile teams is immense (P1, P4-P10, P12-P20, P22-P23, P25-27, P29, P31, P33-35, P39-41, P43, P52-53, P55). The success of Agile adoption, and that of the self-organizing Agile teams, is dependent on senior management support. The *Champion* role emerged on Agile teams to secure senior management support [15]. Fig. 7 illustrates how the category *Champion* emerged from the underlying concepts.

> "*...the organizations I see getting the most benefit from Scrum, from Agile, are organizations where senior management really gets it! Where senior management has been has been through training...Senior management took the time to read, learn about Agile. The least successful Agile adoptions are ones where senior management has no interest in Agile, they have no interest in what Agile is.*"—P43, Scrum Trainer, India

While senior management support is extremely important for self-organizing Agile teams, it doesn't always come naturally. Supporting these teams involves the senior management changing their organizational culture, process of negotiating contracts, and resource management strategies. All senior management may not be ready to make such significant, organization-wide changes.

> "*...main problem is, out of ten, nine people are agreeing to do [Agile] and one [is] not, and that one is on higher authority...that's a problem...that's a problem!*"—P39, Agile Coach, India

4.4.1 Securing Senior Management Support

A *Champion* is able to **understand the business drivers** (factors that motivate business decisions) that motivate senior management, such as cost effectiveness, time to market, customer demands, and process improvement. The *Champion* **convinces senior management** while keeping in mind these drivers in order to gain their support for the self-organizing Agile team.

> "*For a couple of years now I've been involved within our company to promote this notion...we finally got the okay, a couple of weeks back, to go ahead and make it all formal. Which is excellent, but it took a hell of a long time to understand people's motivations and awareness of things...If you manage to understand their perspective, their buttons, what matters to them, what brings them their next bonus, and paint it in those terms: look, we have just the solution, sign here!*"—P20, Agile Coach, New Zealand

From a senior management perspective, Agile methods are one of several methods from a toolset that their teams can learn and use to better serve their customers. The applicability of a given method to a given project context is an important driver for senior management. Senior management typically remains open to various options that will bring good returns on investment:

> "*To be honest I was doubtful that it was an appropriate type of project to use Agile for, because in my mind it's most useful where there's a lot of user interaction ... I think what you need to understand is the applicability in certain situations—what risks and benefits you're likely to get from different methods at different points and be able to question the approach that's being used*"—P18, Senior Management, NZ

A pragmatic *Champion* is aware that Agile methods may not be applicable to all types of project contexts [11], [104]. A *Champion* is cautious not to advocate Agile irrespective of project context—an effort that can eventually backfire.

> "*...recognizing that Agile does not deliver to every type of project. So for example ... An infrastructure project, I don't think you could put the servers floating in mid air, before you put the wiring in, see if it fit, you know, so things like that.*"—P7, Agile Coach, NZ

And so a *Champion* can advocate the use of Agile methods based on their applicability to the project context. They explain the advantages of Agile methods, given the organization's context. Senior management is much more likely to be convinced to invest in self-organizing Agile teams if they find that the practices fit the projects' contexts.

In order to gain senior management support for exploring Agile methods, a *Champion* **establishes pilot teams**. The idea is to show senior management how Agile practices work on a small scale. Some *Champions* prefer piloting with a team that is open to trying Agile. Most *Champions* mention that the initial pilot attempt works best on a project that had previously experienced difficulties with a traditional development approach so that the value brought in by Agile is more apparent:

> "*Piloting is the key. Pilot with people who want to do it... with a project which has had problems, with changing requirements, with customers not happy. Then you'll see maximum value... if it is a hundred people organization with ten projects, try with one or two [projects].*"—P27, Developer, India

The role of the *Champion* is to educate senior management about Agile methods and the importance of their role in establishing and nurturing self-organizing Agile teams.

> "*You have to recognize that executives are not the enemy; they're you're best allies. They have an intense interest in the organization's success; they're not the ones who prevent you from doing stuff, they just don't know any better! (laughs) So if you see them as misinformed people...they're victims of the current mindset. The only thing you can do is recognize them as such and treat them as such. Educate them, gently.*"—P20, Agile Coach, New Zealand

A team is impacted in several ways by the senior management at their own organization: Senior management influences organizational culture, types of contracts governing projects, financial sponsorship, and resource management. A lack of understanding of Agile principles and practices can lead senior management to take project decisions that can adversely affect the self-organizing ability of the Agile team.

4.4.2 Propagating More Teams

The role of the *Champion* is not limited to driving initial pilot projects. The *Champion* also promotes the idea of propagating more self-organizing Agile teams across the organization.

In the absence of any real drivers for change, however, senior management are not convinced about adopting Agile methods and making the organization-wide changes required to support self-organizing Agile teams. One example from this research study is that of an organization where a pilot team had become a high performing and self-organizing Agile team. The project, however, was ultimately brought to an end by senior management as a part of a restructuring effort in response to a global economic recession. The senior management, by that time, had not seen any real reason to invest in self-organizing Agile teams, especially in the face of an economic crisis.

> "*They [senior management] scattered the one effective Agile team to the four winds—one of the coders went back to website content and all the other BAs have been re-assigned or let go...so I just don't know how big a priority Agile was at the time. I really believe it should be our standard methodology; I drink Kool-aid, I'm converted! I think it's the way to go and I just don't know how receptive the business was at the time, or whether it was just 'we've got to save money'.*"—P9, Customer Representative, New Zealand

The *Champion* role was played mostly by Agile coaches and by a developer in one case. Once the senior management is convinced that Agile software development is advantageous to their organization, the senior management may take over the role of *Champion* and champion the cause of propagating self-organizing Agile teams across the organization. The senior management, in the role of *Champion*, influences organizational culture, types of contracts governing projects, financial sponsorship, and resource management to favor proper functioning self-organizing Agile teams.

4.5 Promoter

Promotes Agile with customers in an attempt to secure their involvement and collaboration to support the self-organizing Agile team.

Besides senior management support, another critical environmental factor that influences self-organizing Agile teams is the level of customer involvement. Inadequate customer involvement is a common challenge that many Agile teams face (P1-P2, P4, P5-P9, P11-P14, P19-20, P25, P27, P43, P54). Inadequate customer involvement causes several challenges for the self-organizing Agile team, such as problems in gathering and clarifying requirements, problems in prioritization and receiving feedback, productivity loss, and even business loss in some cases. There are several causes leading to inadequate customer involvement. These include skepticism among customers, distance between customers and the team, lack of time commitment on the part of the customers, etc. The *Promoter* role emerged to overcome the lack of customer involvement [10], [12]. Fig. 8 illustrates how the category *Promoter* emerged from the underlying concepts.

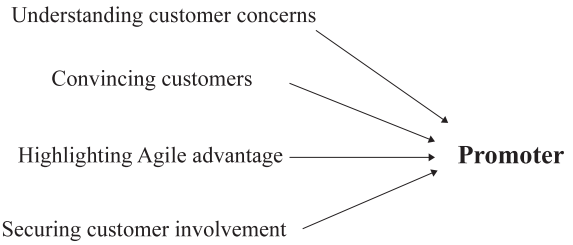

Fig. 8. Emergence of the category *Promoter* from underlying concepts.

4.5.1 Understanding Customer Concerns

Customers can harbor misconceptions and skepticism about Agile software development. As one of the customer representatives disclosed, they were extremely skeptical about Agile methods at the beginning of the project:

> "*I remember is someone was talking to me—and I knew nothing about Agile so it was like what the hell is Agile?—and I got a brief overview and I though that seems remarkably sensible, the basic principles. And then...all I know is someone came up to me very excitedly and 'oh we've got a scrum coach coming in this week!' Are we playing rugby?! Is there a social team? I used to play a lot, I could come in handy! And they're like 'no, it's Agile' and I was like what is Scrum and why do you need a coach?*"—P9, Customer Representative, New Zealand

Part of the *Promoter*'s role is to understand the customer's background in terms of their understanding of Agile methods and consequently their readiness for collaboration with the team. A *Promoter* tries to understand the concerns of their customers before advocating the use of Agile methods.

> "*Agile has been there for a while, people are waking up to this concept [now]. This huge hullabaloo about Agile this, Agile that! we showcase our [unique] offering, we showcase case studies and also give them a sense of—not lulling them into a sense of security but—real values and also focusing on the hardship which comes with that...*" — P36, Agile Coach, India

4.5.2 Securing Customer Involvement

The collaboration between the team and customers ensures the development of a product that is built to the customer's vision. Convincing the customer that this advantage is worth their time and securing their collaboration is challenging [12]. Customers may not realize their responsibilities on an Agile project:

> "*The client reads [Scrum books] and what they see is client can make changes all the time and they think wow that sounds great!... They don't understand the counter-balancing discipline [customer involvement] ... Customer involvement is poor.*"—P43, Scrum Trainer, India

The *Promoter* identifies the concerns of the customers, and systematically attempts to engage them with Agile practices.

> "*I did persuade the client to go down this road...story cards, iterations, all the way through. Slowly the client did come around and started to see benefit, so it did work out really well*"—P19, Senior Management/Agile Coach, NZ

One of the ways a *Promoter* attempts to **convince customers is by highlighting the advantages of Agile software development**. Customer involvement in the project helps the team to avoid rework:

> "*To get the client involved in the process I think is the most difficult part of Agile...[customer involvement is a] benefit for us [team], because we don't have to redo things. So from my perspective as a developer, yes, the more the client is involved, the better for us.*"—P14, Developer, New Zealand

In the absence of a customer who understands Agile methods and is willing to collaborate, a self-organizing team is unable to function to its full potential.

> "*Two of the [internal customers] responded lots and were very... complaining, and at the end of the project their business units loved it and the business unit that didn't give much feedback — when it went to a user — started complaining. And it's like well, if we didn't get any critique it's not really our fault!*"—P11, Developer, New Zealand

Given the collaboration-intensive nature of Agile practices, a self-organizing Agile team cannot work and flourish in isolation. The *Champion* and *Promoter* roles were crucial in identifying the influence of the environmental factors—support of senior management and customer involvement—and securing their support, respectively. In new teams, these roles were usually played by Agile coaches. In more mature teams, any experienced team member can play these roles, embodied by the same person in some cases.

Both the *Promoter* and *Coordinator* roles are customer focused. The *Promoter* attempts to secure adequate levels of customer involvement on the project for the proper functioning of the team. The *Coordinator* role emerges in situations where the level of customer involvement is inadequate despite the *Promoter*'s attempts to secure involvement. In contrast, the *Champion* attempts to secure senior management support for the team. If the *Champion* fails, the future of the self-organizing team is seriously jeopardized. This suggests that while adequate customer involvement is highly beneficial for a self-organizing Agile team, senior management support is imperative.

4.6 Terminator

Identifies team members threatening the proper functioning and productivity of the self-organizing Agile team and engages senior management support in removing such members from the team.

Self-organizing Agile teams are "*open*" in nature and willing to "*change*" (P1, P5, P7, P9, P10, P12-P14, P18, P20, P26-P29, P31, P36, P47-52). In the absence of these desired characteristics, the individual is perceived to pose a threat to the proper functioning and productivity of the self-organizing Agile team. The *Terminator* role emerged to identify team members threatening the proper functioning of the self-organizing Agile team, and to seek senior management support in removing such members. The role of the *Terminator* is certainly not an easy one, and is perhaps the most controversial. Fig. 9 illustrates how the category *Terminator* emerged from the underlying concepts.

Fig. 9. Emergence of the category *Terminator* from underlying concepts.

4.6.1 Identifying Threatening Team Members

The *Terminator* identifies individuals in a team who may be hampering team productivity because of their personal characteristics and practices. The individual personalities of team members can be considered more important than the skill set when selecting an Agile team. As one of the *Terminators* acknowledges below, the individuals themselves are not "*bad*," but that their personalities are not suited to the Agile way of working, which starts to hamper the productivity of the entire team. Removing such team members who hamper the team's productivity can be crucial to project success:

> "*If you have someone who isn't willing to learn and just communicate—all those kind of key things that are needed in an Agile team member—they can wreck the project very very quickly. Your only tester who refuses to adjust the process to fit the speed of the team is dogmatic about the way they work or a developer who doesn't like communicating, wants to keep their head down on the computer doesn't like to talk to people when they have a problem and instead try and solve it themselves and the whole team can go—as soon as one story is overdue and out of whack it can be critical path in no time flat because you are doing this just in time...It's the whole team, it doesn't matter. It is the project manager or the tester or the BA or the developers themselves. Any one of them that can't adjust to the Agile mechanism really needs to be removed pretty quickly...The faster you sort out the bad elements, the better. It's not that the person is bad, they may be very very good at their job, it's just that they can't adjust to the different mechanism [of working].*"—P10, Agile Coach, NZ

While inability to adjust to the Agile way of working is seen as a disadvantage by many *Terminators*, the other extreme of embodying idealistic or evangelist attitude toward Agile software development is also seen as a potential hindrance to the self-organization in an Agile team:

> "*Some evangelists have such hundred percent concepts—just scares me as a coach...Throw out evangelists sometimes, hard reality! People get fired. It's the cold-hearted nature of this businesses, [Agile] identifies the good things, [Agile] identifies even the bad things. Sometimes [we] have to throw people out.*"—P36, Agile coach, India

The required characteristics of individuals on Agile teams include openness, communication, ability to change,

and ability to learn. The difference between members of self-organizing Agile teams and those from traditional teams is so apparent that it doesn't escape the notice of senior management.

> "*I think the personal interactions and behaviours of the group is interesting in its own way; they're more communicative with people. You dealing with people who are positively more social I dont think thats just because of the people who were chosen, they seem to be more social and communicative generally. The people working on non-Agile projects tend to be very isolated in terms of their behaviours they're not actually isolated, they could talk to people, but they don't tend to so much.*"—P18, Senior Management, NZ

4.6.2 Removing Members from the Team

Sometimes a team member can destabilize the team by their actions and even though the other team members are aware of it, they are unable to express their concerns. The *Terminator* identifies the latent concerns of the rest of the team and **seeks senior management support** in removing such members:

> "*[Everything] seemed to go all right until [team member] tore the whole product apart...So our [Terminator] came in...noted that [team member] was holding the team back, and made an executive decision by talking to management as the [Terminator] and said 'the Agile method isn't working in this team because this one person is making such a large difference to everyone's productivity'...[we] simply didn't want to voice our opinions because there was too much fall-back when we tried to...But the [Terminator] really made that quite obvious to management and therefore we [the organization] just removed them.*"—P4, Business Analyst, NZ

> "*We had two BAs and they just wouldn't get it because they had been working on 'going away with your specs, and then come back' and I had a mandate to actually pull out those people who were not working, um I had both of them boarded off!*"—P23, Agile Coach, NZ

Selecting members up front is one of the activities a *Terminator* engages in. In mature teams, the whole team provides input in the hiring process, which influences the *Terminator* to select individuals upfront.

> "*[At the time of hiring] it was just 'well who is going to work better with this group of people?' rather than who's better technically or anything...[The team] came down to the point where they're [a couple of applicants] both equal and then personality's more important so we have a couple of us just figure out who we want to work with more. But I think that's really important with Agile; you've got to have people you can work that closely with and trust, a lot more than if you're doing Waterfall*"—P11, Developer, NZ

The *Terminator* role was played by experienced Agile coaches in new teams. The *Terminator* was played by an Agile coach supported by the whole team in mature Agile teams. In Agile organizations, the role of the *Terminator* was extended to cover organization-wide issues (P34, P36, P52-53). The organization-wide *Terminator* was played by the HR—Human Resources—Department within the organization. The organization-wide *Terminator* selected new members during the hiring process based on their ability to fit into the self-organizing team culture (P10, P52).

> "*we see when we do a code pairing how this guy [potential recruit] is interacting and how open he is to the idea...So how interactive he is, how he listens to the people and understands the team, and probably explain things back to them to make it come to a smart solution...we find out his cultural fit...[has to be] open for the feedback.*"—P52, Human Resources Manager, India

Both the *Champion* and *Terminator* roles interacted with the senior management. The *Champion* role involved informing senior management about the advantages of Agile software development specific to their context and securing their financial and structural support for enabling changes in organizational culture, contract negotiation, and resource management on an organization-wide level. The *Terminator* role involved identifying team members threatening the proper functioning of the team and seeking senior management support in removing such members on a team level. The two roles were played by the same person in some cases.

5 Discussion

The self-organizing roles were such that one person could play multiple roles at different times, depending on the need of the hour. For example, a developer was playing *Coordinator* and *Translator* roles with customers that were new to Agile and were not providing adequate levels of involvement. The same developer was playing a *Champion* role convincing their senior management to invest in their Agile team. This further confirms the transient, informal, and spontaneous nature of these roles which are hallmarks of self-organizing teams (from a complex-adaptive perspective [66].)

The rest of this section discusses the research results in light of related work, presents the implications for practice, outlines the criteria used for evaluating a grounded theory, describes the limitations and threats to validity, and finally, presents some ideas for future work.

5.1 Related Work

Following Grounded Theory, the data were first collected and then analyzed. Once the findings were sufficiently grounded and developed, the literature on self-organizing Agile teams was reviewed. The purpose of literature review after analysis is 1) to protect the findings from preconceived notions and 2) to relate the research findings to the literature through integration of ideas [78]. This section discusses our results in the light of the related literature.

A wide number of researchers have explored team roles and dynamics [17], [16], [3], [105], [51], [106].

Belbin suggests nine team roles based on behavior: plant, resource investigator, coordinator, shaper, monitor evaluator, teamworker, implementer, completer finisher, and specialist [16]. A coordinator in Belbin's team roles theory focuses on team's objectives and delegates work. The *Coordinator* role identified in our research, on the other hand, helps coordinate between the team and their customers and does not delegate work. A key practice of self-organizing Agile teams is self-assignment. A specialist in Belbin's theory focuses on a particular area of expertise and has a tendency to value their specialization over team goals. In self-organizing Agile teams, however, team members balance between cross functionality and specialization while remaining committed to the team goal.

Five boundary-spanning roles have been identified as means to encourage communication across boundaries: ambassador, scout, guard, sentry, and coordinator [17],

[106]. An ambassador represents the team to external stakeholders and persuades them to support the team. This is similar to the *Champion* and *Promoter* roles identified in our research, where the *Champion* persuades senior management to support the team and the *Promoter* persuades customers to support the team through collaboration. A scout is responsible for scanning within and outside their organizations for new ideas and technologies. In self-organizing Agile teams, on the other hand, learning new technologies and concepts is a continuous effort performed by all team members. The guard and the sentry roles are meant to protect the team from external distractions and act as filters, regulating the information passing into and out of the team. Our research did not identify such roles on self-organizing Agile teams. Instead of taking on defensive roles (such as guard and sentry), self-organizing Agile teams proactively seek the support of their environmental factors through *Champion* and *Promoter* roles. The last of the five boundary-spanning roles is the coordinator. Much like the *Coordinator* role identified in our research, this coordinator role focuses on communication with external groups while keeping them informed of the team's progress.

Anderson and McMillan [66] define self-organizing teams as teams that are

1. informal and temporary,
2. formed spontaneously around issues
3. are not a part of a formal organization structure,
4. possessing a strong sense of shared purpose,
5. where team members decide their own affairs, and
6. where all members' primary roles relates to the task.

The roles identified in this research (*Mentor*, *Coordinator*, etc.) fit each of these criteria of self-organizing teams. Specifically, these roles display the characteristics of self-organizing teams such as being informal, temporary, and formed spontaneously around issues [66]. In other words, these informal, implicit, transient, and spontaneous roles make Agile teams self-organizing [15].

An Agile environment of working is marked by free flow of information and high levels of transparency. For example, various metrics and status of team progress are made highly visible.

Software development teams benefit from the initial guidance of a full-time *Mentor*, played by an experienced Agile coach. Another Grounded Theory study also concluded that a mentor is extremely important in helping newcomers on a project feel better oriented and settle-in [105]. Our *Mentor* role is the closest to the classic Agile coach described in the Agile literature [9], [107], [108], [109].

Some studies have described individuals supporting customers by translating *technical* langauge to *business* language [40], [49]. In contrast, our *Translator* role was able to achieve two-way communication between the development team and their customers by translating *business* language into *technical* language and vice versa. Another difference is that the *Translator* interacted directly with both parties and was a part of the development team. The *Translator* role was played by potentially anyone and everyone on the team.

Cockburn and Highsmith [3] recommend placing "*more emphasis on people factors in the project: amicability, talent, skill, and communication.*" In our research, practitioners used their own set of criteria to evaluate how well an individual fits into an Agile environment, such as communication, ability to give and take feedback, and openness. The *Terminator* exercises their power when team members do not fit in with the rest of the team and hamper their productivity due to lack of openness and willingness to change.

5.2 Implications for Practice

The transition of becoming a self-organizing Agile team is not easy. The roles described in this paper should help team members understand their roles and practices when becoming a self-organizing Agile team.

One of the characteristics of self-organizing teams is their ability to react spontaneously in response to challenges. In an Agile environment, teams can expect to get involved in a lot more practices than just coding and testing. These practices include group programming (as compared to working in isolation), daily stand-ups meetings, and the use of information radiators to promote transparency, collective decision making, and self-assignment.

In the absence of a manager that handles external relations for the team, team members should be ready to take on the interfacing roles of *Coordinator* and *Translator*. Initially, individuals with good communication skills will find themselves taking on these roles. Similarly, team members should be ready to champion their teams with senior management or promote their teams with customers as required by playing *Champion* and *Promoter* roles, respectively. Over time, all members can expect to take up any or all of these team roles as needed.

While some members of the team may easily adjust to the new environment made up of these roles and practices, others may struggle, and some may fail to make the transition. Those members that struggle should try to identify and address pain areas with the help of their *Mentors*. Those individuals who are unable to fit into an Agile way of working may eventually be removed from the team by a *Terminator*.

The popularity of Agile methods has led to several project managers from traditional software development backgrounds taking on an Agile coach role. A new Agile coach often finds themself confused about their role on a self-organizing Agile team. They may be unsure about the level of involvement expected of them. The self-organizing Agile team roles described in this paper should help Agile coaches better understand the responsibilities they are likely to take on at the different stages of the team's maturation and should assist Agile coaches in guiding their team into self-organization. An important implication for the Agile coach, however, is to always be mindful of the self-organizing nature of these roles and facilitate their emergence rather than forcing them on the team.

5.3 Evaluating a Grounded Theory

Glaser recommends that a grounded theory[1] should be evaluated on the basis of four criteria: fit, work, relevance, and modifiability [76].

Fit refers to "*the ability of the categories and their properties to fit the realities under study in the eyes of the subjects, practitioners, and researchers in the area*" [97]. In other words, an emerging theory is said to "fit" if it explains and fits the

1. Grounded Theory is used to refer to the research method, while grounded theory (lower case) is used to refer to the product of the research.

experiences of participants as well as different practitioners who were not involved in theory generation [87]. Publications based on the emerging theory were shared with the participants, many of whom found them relevant and useful. For example, one of the participants provided their feedback via e-mail on the emerging theory as follows:

> *"These [publications] all look very good! The content of all three would be quite useful to members of our organization as well as perhaps our clients."*—P23, Senior Management, New Zealand

The emerging theory was presented to several practitioner groups in India and New Zealand. Confidence in the validity of the emerging theory was helped by these practitioner groups recognizing their own experiences in theory generated from others' experiences.

Work refers to "*the ability of the theory to explain the major variations in behaviour in the area with respect to the processing of the main concerns of the subjects.*" The emerging codes, concepts, and categories were strongly related to the main concern of the participants—becoming a self-organizing Agile team. Frequent presentations to (and feedback from) the Agile practitioner communities in New Zealand and India as well as frequent discussions with the research supervisors about emerging codes, concepts, and categories helped ensure that the emerging theory works.

Relevance is achieved when the criteria of fit and work are met. Relevance evokes instant "*grab*" [97]. Relevance of the emerging theory was established via feedback from practitioners and international experts. Presentations were made at various Agile practitioner group events and to software engineering experts at major international conferences to gain their feedback [89], [15], [104], [11], [10]. When the experts in the field find the research findings useful, it becomes an important source of verifying the fit, work, and relevance of the theory [58]. Receiving comments such as "*well applied*," "*rings true*," and "*I could identify each of those roles*" from the expert reviewers and Agile practitioners made us confident of our emerging theory. Examples of our emerging theory being found relevant include a number of articles by Agile practitioners dedicated to our research [110], [111], [112], [113], [114].

Modifiability is a "*quality of the theory to be ready for changes to include variations in emergent properties and categories caused by new data*" [97]. The emerging theory was modifiable throughout the research. For example, the self-organizing roles evolved through the research as we went from studying relatively new teams to more mature Agile teams.

In addition to these criteria, the ability of a theory to fit and extend the previous literature on the subject also helps evaluate it. Since the major literature review in the same substantive area of research is conducted only after the main concepts and categories are established, the literature becomes an important source of validating the emerging theory. We have discussed how our findings of the self-organizing team roles related to the previous literature in Section 5.1.

5.4 Limitations and Threats to Validity

A limitation of this research study is that the contexts studied were dictated by the choice of research destinations, which in turn were in some ways limited by our access to them. Similarly, the selection of research participants was limited by their willingness to participate.

As with any empirical software engineering, the very high number of variables that affect a real software engineering project make it difficult to identify the impact that any one factor has on the success or failure of the project. The self-organizational roles, practices, and factors influencing self-organizing Agile teams, however, were clearly evident.

A Grounded Theory research study produces a "mid-ranged" theory, which means that while the theory is not claimed to be universally applicable, it can be modified by constant comparison to accommodate more data from new contexts [76]. A key contribution of a GT study, carried out correctly, is that it focuses on conceptualization and produces flexible, modifiable concepts with "*immense grab*" [115]. These concepts interrelate to generate an abstract theory which explains the main concerns of the participants in a substantive area.

The grounded theory of self-organizing Agile teams generated in this research is a first of its kind in the field. Further research into self-organizing teams in Agile software development will help generate a more generalized theory.

5.5 Future Work

Self-organizing Agile team roles ensure that a single team is able to achieve and sustain self-organization by catering to the different needs of the team, such as the need for training, senior management support, customer involvement, etc.

In Agile organizations where all software development is done by multiple self-organizing Agile teams, the self-organizational roles at the team level need organization-wide counterparts at the organizational level. Since all teams are self-organizing, the need for mentoring, training, securing, and coordinating customers collaboration, and removing cultural misfits becomes organization-wide concerns. In response, the self-organizational team roles of *Mentor*, *Coordinator*, *Translator*, *Champion*, *Promoter*, and *Terminator* can be mirrored at the organizational level. The presence of these organization-wide roles were indicated in two mature Agile organizations toward the end of this research. Future work could study Agile software development companies to explore such organization-wide roles that enable self-organization at an organizational level.

6 Conclusion

Self-organizing teams are at the heart of Agile software development. The lack of research into self-organizing teams in software engineering means there is little empirical evidence to explain how Agile teams organize themselves in practice. Through a Grounded Theory study involving 58 Agile practitioners from 23 software organizations in New Zealand and India conducted over a period of four years, we identified six informal, implicit, transient, and spontaneous roles on Agile teams that make them self-organizing. These roles are:

1. Mentor, who guides and supports the team initially, helps them become confident in their use of Agile methods, ensures continued adherence to Agile methods, and encourages the development of self-organizing practices in the team;
2. Coordinator, who acts as a representative of the team to manage customer expectations and coordinate customer collaboration with the team;

3. Translator, who understands and translates between the business language used by customers and the technical terminology used by the team to improve communication between the two;
4. Champion, who champions the Agile cause with the senior management within their organization in order to gain support for the self-organizing Agile team;
5. Promoter, who promotes Agile with customers and attempts to secure their involvement and collaboration to support the efficient functioning of the self-organizing Agile team; and
6. Terminator, who identifies team members threatening the proper functioning and productivity of the self-organizing Agile team and engages senior management support in removing such members from the team.

We also presented a detailed review of self-organizing teams from multiple perspectives and discussed the implications of our findings for practitioners. Understanding these roles will help software development teams and their managers better comprehend and execute their roles and responsibilities as a self-organizing team.

ACKNOWLEDGMENTS

The authors would like to thank BuildIT (NZ), SPPI (NZ), Agile Alliance (USA), and all the participants for their support of this research.

REFERENCES

[1] J. Highsmith and M. Fowler, "The Agile Manifesto," *Software Development Magazine,* vol. 9, no. 8, pp. 29-30, 2001.
[2] T. Chow and D. Cao, "A Survey Study of Critical Success Factors in Agile Software Projects," *J. Systems and Software,* vol. 81, no. 6, pp. 961-971, 2008.
[3] A. Cockburn and J. Highsmith, "Agile Software Development: The People Factor," *Computer,* vol. 34, no. 11, pp. 131-133, Nov. 2001.
[4] R. Martin, *Agile Software Development: Principles, Patterns, and Practices.* Pearson Education, 2002.
[5] K. Schwaber, "Scrum Guide," Scrum Alliance Resources, http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf, Nov. 2010.
[6] H. Sharp and H. Robinson, "An Ethnographic Study of XP Practice," *Empirical Software Eng.,* vol. 9, no. 4, pp. 353-375, 2004.
[7] J. Highsmith, *Agile Project Management: Creating Innovative Products.* Addison-Weasley, 2004.
[8] XP, "Extreme Programming: A Gentle Introduction," World Wide Web electronic publication, http://www.extremeprogramming.org/, Oct. 2010.
[9] S. Augustine, *Managing Agile Projects.* Prentice Hall PTR, 2005.
[10] R. Hoda, J. Noble, and S. Marshall, "Agile Undercover: When Customers Don't Collaborate," *Proc. Int'l Conf. Agile Processes in Software Eng. and Extreme Programming,* pp. 73-87, 2010.
[11] R. Hoda, J. Noble, and S. Marshall, "Supporting Self-Organizing Agile Teams—What's Senior Management Got to Do with It?" *Proc. Int'l Conf. Agile Processes in Software Eng. and Extreme Programming,* pp. 73-87, 2011.
[12] R. Hoda, J. Noble, and S. Marshall, "Impact of Inadequate Customer Involvement on Self-Organizing Agile Teams," *J. Information and Software Technology,* vol. 53, no. 5, pp. 521-534, 2011.
[13] R. Hoda, "Self-Organizing Agile Teams: A Grounded Theory," PhD dissertation, Eng. and Computer Science, Victoria Univ. of Wellington, New Zealand, 2011.
[14] R. Hoda, J. Noble, and S. Marshall, "Developing a Grounded Theory to Explain the Practices of Self-Organizing Agile Teams," *Empirical Software Eng. J.,* in press, 2011.
[15] R. Hoda, J. Noble, and S. Marshall, "Organizing Self-Organizing Teams," *Proc. 32nd ACM/IEEE Int'l Conf. Software Eng.,* pp. 285-294, 2010.
[16] R. Belbin, *Team Roles at Work.* Butterworth-Heinemann, 1993.
[17] D.G. Ancona and D.F. Caldwell, "Beyond Task and Maintenance: Defining External Functions in Groups," *Group Organization Management,* vol. 13, no. 4, pp. 468-494, 1988.
[18] C. Larman and V.R. Basili, "Iterative and Incremental Development: A Brief History," *Computer,* vol. 36, no. 6, pp. 47-56, June 2003.
[19] M. Lindvall, V.R. Basili, B.W. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L.A. Williams, and M.V. Zelkowitz, "Empirical Findings in Agile Methods," *Proc. Second XP Universe and First Agile Universe Conf. Extreme Programming and Agile Methods,* pp. 197-207, 2002.
[20] D. Robey, R. Welke, and D. Turk, "Traditional, Iterative, and Component-Based Development: A Social Analysis of Software Development Paradigms," *Information Technology and Management,* vol. 2, no. 1, pp. 53-70, 2001.
[21] J. Stapleton, *Dynamic Systems Development Method.* Addison Wesley, 1997.
[22] P. Abrahamsson, *Agile Software Development Methods: Review and Analysis.* VTT Publications, 2002.
[23] A. Cockburn, *Crystal Clear: A Human-Powered Methodology for Small Teams.* Addison-Wesley Professional, 2004.
[24] S. Palmer and M. Felsing, *A Practical Guide to Feature-Driven Development.* Pearson Education, 2001.
[25] J. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* Dorset House Publishing, 2000.
[26] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still, "The Impact of Agile Practices on Communication in Software Development," *Empirical Software Eng.,* vol. 13, no. 3, pp. 303-337, 2008.
[27] Scrum Alliance, World Wide Web electronic publication, http://www.scrumalliance.org/view/scrum_framework, Sept. 2008.
[28] H. Takeuchi and I. Nonaka, "The New New Product Development Game," *Harvard Business Rev.,* vol. 64, no. 1, pp. 137-146, 1986.
[29] Control Chaos, World Wide Web electronic publication, http://www.controlchaos.com/old-site/rules.htm, Sept. 2010.
[30] K. Beck, *Extreme Programming Explained: Embrace Change,* first ed. Addison-Wesley Professional, 1999.
[31] P. Abrahamsson, J. Warsta, M.T. Siponen, and J. Ronkainen, "New Directions on Agile Methods: A Comparative Analysis," *Proc. 25th Int'l Conf. Software Eng.,* pp. 244-254, 2003.
[32] T. Dybå and T. Dingsoyr, "Empirical Studies of Agile Software Development: A Systematic Review," *Information Software Technology,* vol. 50, nos. 9/10, pp. 833-859, 2008.
[33] J.E. Tomayko and O. Hazzan, *Human Aspects of Software Engineering.* Charles River Media, 2004.
[34] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of Migrating to Agile Methodologies," *Comm. ACM,* vol. 48, no. 5, pp. 72-78, 2005.
[35] S. Nerur and V. Balijepally, "Theoretical Reflections on Agile Development Methodologies," *Comm. ACM,* vol. 50, no. 3, pp. 79-83, 2007.
[36] F.P. Brooks, *Mythical Man-Month,* second ed. Addison-Welsey, 1995.
[37] H.D. Mills, *Software Productivity.* Little Brown and Company, 1983.
[38] P.S. Taylor, D. Greer, P. Sage, G. Coleman, K. McDaid, and F. Keenan, "Do Agile GSD Experience Reports Help the Practitioner?" *Proc. Int'l Workshop Global Software Development for the Practitioner,* pp. 87-93, 2006.
[39] A. Cockburn, "People and Methodologies in Software Development," PhD dissertation, Univ. of Oslo, Norway, 2003.
[40] C. Mann and F. Maurer, "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction," *Proc. Agile Development Conf.,* pp. 70-79, 2005.
[41] E. Whitworth and R. Biddle, "The Social Nature of Agile Teams," *Proc. Agile '07,* pp. 26-36, 2007.
[42] L. Williams, R.R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software,* vol. 17, no. 4, pp. 19-25, July/Aug. 2000.
[43] A. Begel and N. Nagappan, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," *Proc. First Int'l Symp. Empirical Software Eng. and Measurement,* pp. 255-264, 2007.

[44] H. Sharp and H. Robinson, "Collaboration and Co-Ordination in Mature Extreme Programming Teams," *Int'l J. Human-Computer Studies,* vol. 66, no. 7, pp. 506-518, 2008.

[45] H. Robinson and H. Sharp, "Organisational Culture and XP: Three Case Studies," *Proc. Agile Development Conf./Australasian Database Conf.,* pp. 49-58, 2005.

[46] H. Robinson and H. Sharp, "The Social Side of Technical Practices," *Proc. Sixth Int'l Conf. Extreme Programming and Agile Processes in Software Eng.,* pp. 100-108, 2005.

[47] L.A. Williams and R.R. Kessler, "All I Really Need to Know about Pair Programming I Learned in Kindergarten," *Comm. ACM,* vol. 43, no. 5, pp. 108-114, 2000.

[48] A. Martin, R. Biddle, and J. Noble, "The XP Customer Role in Practice: Three Studies," *Proc. Agile Development Conf.,* pp. 42-54, 2004.

[49] A. Martin, R. Biddle, and J. Noble, "The XP Customer Role: A Grounded Theory," *Proc. AGILE '09,* 2009.

[50] P. Sfetsos, L. Angelis, and I. Stamelos, "Investigating the Extreme Programming System—An Empirical Study," *Empirical Software Eng.,* vol. 11, no. 2, pp. 269-301, 2006.

[51] N.B. Moe, T. Dingsoyr, and T. Dybå, "Understanding Self-Organizing Teams in Agile Software Development," *Proc. 19th Australian Conf. Software Eng.,* pp. 76-85, 2008.

[52] E. Trist, "The Evolution of Socio-Technical Systems," *Occasional Paper,* 1981.

[53] J. Baker, "Tightening the Iron Cage: Concertive Control in Self-Managing Teams," *Administrative Science Quarterly,* vol. 38, no. 3, pp. 408-437, 1993.

[54] K. Lewin, *Resolving Social Conflicts: Selected Papers on Group Dynamics.* Harper and Row, 1948.

[55] K. Schwaber, "Agile Processes and Self-Organization," Control Chaos, http://www.controlchaos.com/download/SelfSelf%20Organization.pdf, Mar. 2010.

[56] J. Hut and E. Molleman, "Empowerment and Team Development," *Team Performance Management,* vol. 4, no. 2, pp. 53-66, 1998.

[57] E. Molleman, "Variety and the Requisite of Self-Organization," *Int'l J. Organizational Analysis,* vol. 6, no. 2, pp. 109-131, 1998.

[58] G. Morgan, *Images of Organization.* Sage Publications, 1986.

[59] "Book Reviews Comptes Rendus," *Canadian Public Administration,* vol. 32, no. 2, pp. 320-339, 1989.

[60] R. Ashby, *An Introduction to Cybernetics.* Chapman and Hall, 1956.

[61] I. Nonaka, "A Dynamic Theory of Organizational Knowledge Creation," *Organization Science,* vol. 5, no. 1, pp. 14-37, 1994.

[62] S.A. Kauffman, *The Origins of Order.* Oxford Univ. Press, 1993.

[63] S.J. Lansing, "Complex Adaptive Systems," *Ann. Rev. Anthropology,* vol. 32, pp. 183-204, 2003.

[64] S.A. Levin, "Ecosystems and the Biosphere as Complex Adaptive Systems," *Ecosystems,* vol. 1, no. 5, pp. 431-436, 1998.

[65] R. Lewin, "From Chaos to Complexity: Implications for Organizations," *Executive Development,* vol. 7, no. 4, pp. 16-17, 1994.

[66] C. Anderson and E. McMillan, "Of Ants and Men: Self-Organized Teams in Human and Insect Organizations," *Emergence: Complexity Organization,* vol. 5, no. 2, pp. 29-41, 2003.

[67] R. Lewin, *Complexity—Life at the Edge of Chaos.* Dent, 1993.

[68] J. Sutherland, "Roots of Scrum: Takeuchi and Self-Organizing Teams," World Wide Web electronic publication, http://jeffsutherland.com, Mar. 2010.

[69] D. Larsen, "Team Agility: Exploring Self-Organizing Software Development Teams," Industrial Logic and The Agile Times Newsletter, http://www.futureworksconsulting.com/resources/TeamAgilityAgileTimesFeb04.pdf, Nov. 2010.

[70] M. Berteig, "Team Self-Organization," Agile Advice, http://www.Agileadvice.com/archives/2005/12/Agile_work_uses_2.html, Nov. 2010.

[71] A. Elssamadisy, *Agile Adoption Patterns: A Roadmap to Organizational Success.* Addison-Weasley Professional, 2008.

[72] T. Chau and F. Maurer, "Knowledge Sharing in Agile Software Teams," *Proc. Logic versus Approximation,* pp. 173-183, 2004.

[73] L. Anderson, G. Alleman, K. Beck, J. Blotner, W. Cunningham, M. Poppendieck, and R. Wirfs-Brock, "Agile Management—An Oxymoron?: Who Needs Managers Anyway?" *Proc. 18th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications,* pp. 275-277, http://doi.acm.org/10.1145/949344.949410, 2003.

[74] N.B. Moe, T. Dingsøyr, and T. Dybå, "A Teamwork Model for Understanding an Agile Team: A Case Study of a Scrum Project," *Information and Software Technology,* vol. 52, no. 5, pp. 480-491, 2010.

[75] T. Dickinson and R. McIntyre, "A Conceptual Framework for Teamwork Measurement," *Team Performance Assessment and Measurement: Theory, Methods, and Applications,* pp. 19-43, Routledge, 1997.

[76] B. Glaser, *Basics of Grounded Theory Analysis: Emergence vs. Forcing.* Sociology Press, 1992.

[77] B. Glaser and A.L. Strauss, *The Discovery of Grounded Theory.* Aldine, 1967.

[78] B. Glaser, *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory.* Sociology Press, 1978.

[79] G. Allan, "The Legitimacy of Grounded Theory," *Proc. Fifth European Conf. Business Research Methods,* pp. 1-8, 2006.

[80] J.W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods and Approaches,* second ed. Sage Publications, 2003.

[81] K. Charmaz, *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis,* first ed. Sage Publications, 2006.

[82] B. Glaser, "Remodeling Grounded Theory," *Forum: Qualitative Social Research,* vol. 5, no. 2, article 4, 2004.

[83] B. Glaser, "Grounded Theory Institute: Methodology of Barney G. Glaser," http://groundedtheory.org/, Apr. 2010.

[84] S. Adolph, W. Hall, and P. Kruchten, "A Methodological Leg to Stand On: Lessons Learned Using Grounded Theory to Study Software Development," *Proc. Conf. Center for Advanced Studies on Collaborative Research: Meeting of Minds,* pp. 166-178, 2008.

[85] J.Q. Benoliel, "Grounded Theory and Nursing Knowledge," *Qualitative Health Research,* vol. 6, no. 3, pp. 406-428, 1996.

[86] N. Elliot and A. Lazenbatt, "How to Recognize a 'Quality' Grounded Theory Research Study," *Australian J. Advanced Nursing,* vol. 22, no. 3, pp. 48-52, 2005.

[87] A.K. Nathaniel, "A Grounded Theory of Moral Reckoning in Nursing," PhD dissertation, West Virginia Univ., 2003.

[88] A. Lambert, "Fluid Families: A Theoretical Model for Determining Family Membership within Blended and Ex-Blended Families," *Proc. 93rd Ann. Convention Nat'l Comm. Assoc.,* 2007.

[89] APN "Agile Professionals Network," World Wide Web electronic publication, http://www.Agileprofessionals.net/, Sept. 2010.

[90] ASCI, "Agile Software Community of India," World Wide Web electronic publication, http://www.Agileindia.org/, Sept. 2010.

[91] L. Abraham, "Cultural Differences in Software Engineering," *Proc. India Software Eng. Conf.,* pp. 95-100, 2009.

[92] M. Summers, "Insights into an Agile Adventure with Offshore Partners," *Proc. Agile '06,* pp. 333-338, 2008.

[93] K. Sureshchandra and J. Shrinivasavadhani, "Adopting Agile in Distributed Development," *Proc. IEEE Int'l Conf. Global Software Eng.,* pp. 217-221, 2008.

[94] J. Sutherland, G. Schoonheim, E. Rustenburg, and M. Rijk, "Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams," *Proc. Agile '08,* pp. 339-344, 2008.

[95] E. Uy and N. Ioannou, "Growing and Sustaining an Offshore Scrum Engagement," *Proc. Agile '08,* 2008.

[96] B. Glaser, *The Grounded Theory Perspective III: Theoretical Coding.* Sociology Press, 2005.

[97] B. Glaser, *Doing Grounded Theory: Issues and Discussions.* Sociology Press, 1998.

[98] S. Georgieva and G. Allan, "Best Practices in Project Management through a Grounded Theory Lens," *Electronic J. Business Research Methods,* vol. 1, pp. 43-52, 2008.

[99] G. Allan, "A Critique of Using Grounded Theory as a Research Method," *Electronic J. Business Research Methods,* vol. 2, no. 1, pp. 1-10, 2003.

[100] R. Hoda, J. Noble, and S. Marshall, "Balancing Acts: Walking the Agile Tightrope," *Proc. Workshop ICSE Co-Operative and Human Aspects of Software Eng.,* 2010.

[101] S. Fraser, R. Reinitz, J. Eckstein, J. Kerievsky, R. Mee, and M. Poppendieck, "Xtreme Programming and Agile Coaching," *Proc. Companion of the 18th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications,* pp. 265-267, 2003.

[102] P.S. Grisham and D.E. Perry, "Customer Relationships and Extreme Programming," *Proc. Workshop Human and Social Factors of Software Eng.,* pp. 1-6, 2005.

[103] R. Hoda, J. Noble, and S. Marshall, "What Language Does Agile Speak?" *Proc. Int'l Conf. Agile Processes in Software Eng. and Extreme Programming*, pp. 387-388, 2010.
[104] R. Hoda, J. Noble, and S. Marshall, "Agility in Context," *Proc. ACM Int'l Conf. Object Oriented Programming Systems, Languages, and Applications*, pp. 74-88, 2010.
[105] B. Dagenais, H. Ossher, R.K.E. Bellamy, M.P. Robillard, and J.P. de Vries, "Moving into a New Software Project Landscape," *Proc. 32nd ACM/IEEE Int'l Conf. Software Eng.*, pp. 275-284, 2010.
[106] S. Sawyer, P.J. Guinan, and J. Cooprider, "Social Interactions of Information Systems Development Teams: A Performance Perspective," *Information Systems J.*, vol. 20, pp. 81-107, Jan. 2010.
[107] N.B. Moe and T. Dingsoyr, "Scrum and Team Effectiveness: Theory and Practice," *Proc. Int'l Conf. Agile Processes in Software Eng. and Extreme Programming*, pp. 11-20, 2008.
[108] L. Rising and N.S. Janoff, "The Scrum Software Development Process for Small Teams," *IEEE Software*, vol. 17, no. 4, pp. 26-32, July/Aug. 2000.
[109] K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*. Prentice-Hall, 2002.
[110] S. Hastie, "Organizing Self-Organizing Agile Teams," InfoQ article, http://www.infoq.com/news/2010/04/organizing-self-organizing-teams, Nov. 2010.
[111] A. Kearns, "Converting Waterfall Requirements into Underground Agile Features," World Wide Web electronic publication, http://www.morphological.geek.nz/blogs/viewpost/Peruse+Muse+Infuse/Converting+Waterfall+Requirements+into+Underground +Agile+Features.aspx, Nov. 2010.
[112] K.D. Hoeppner, "Agile Undercover," World Wide Web electronic publication, http://virtualbreath.net/curious/2010/08/23/Agile-undercover/, Nov. 2010.
[113] Derby, Esther "A Tale of a Too Hands-Off Manager," World Wide Web electronic publication, http://www.estherderby.com/2010/10/too-hands-off-manager.html, Nov. 2010.
[114] S. Mamoli, "Agile Undercover: When Customers Don't Collaborate," World Wide Web electronic publication, http://www.nomad8.com/files/category-agile-product-ownership.php, Nov. 2010.
[115] B. Glaser, "Naturalist Inquiry and Grounded Theory," *Forum: Qualitative Social Research*, vol. 5, no. 1, article 7, 2004.

Rashina Hoda received the bachelor's degree in computer science summa cum laude from Louisiana State University, the PhD degree in computer science from Victoria University of Wellington, New Zealand, and is a certified Scrum Master. She is a lecturer on software engineering in the Department of Electrical and Computer Engineering, The University of Auckland, New Zealand. Her research interests include software engineering, Agile software development, and human-computer interaction. She is a member of the IEEE.

James Noble is the head of research in the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research centers around software design, ranging from object-orientation, aliasing, design patterns, and Agile methodology, via usability and visualization, to postmodernism and the semiotics of programming. He is a member of the IEEE.

Stuart Marshall is a lecturer in the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include information and software visualization, multitouch user interfaces on mobile devices and table-tops, digital game preservation, and Agile software development. He is a member of the IEEE.