Self-Organizing Agile Teams: A Grounded Theory

by

Rashina Hoda

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2011

Abstract

Self-organizing teams are a hallmark of Agile software development, directly affecting team effectiveness and project success. Agile software development, and in particular the Scrum method, emphasizes self-organizing teams but does not provide clear guidelines on how teams should become and remain self-organizing. Based on Grounded Theory research involving 58 Agile practitioners from 23 different software organizations in New Zealand and India, this thesis presents a grounded theory of self-organizing Agile teams. The 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*, in order to become self-organizing. The roles are: Mentor, Co-ordinator, Translator, Champion, Promoter, and Terminator. The practices involve balancing freedom and responsibility, cross-functionality and specialization, and continuous learning and iteration pressure. The factors are senior management support and level of customer involvement. This thesis will help teams and their coaches better understand their roles and responsibilities as a self-organizing Agile team. This thesis will also serve to educate senior management and customers about the importance of supporting these teams.

Dedication

To Late Mrs. Qamrun Nisa Begam

You inspired me as an academic par excellence, a social worker,
winner of the President's National Award for your life-long contribution
to the cause of girls education in India, and most importantly,
as my grandmother and first teacher.

Acknowledgments

"Is there any Reward for Good—other than Good?
Then which of the favours of your Lord will ye deny?"

– Surah Ar-Rahman, verses 60-61, The Holy Quran

All praise be to God, the Most Gracious, the Most Merciful, for His innumerable favours. Peace and blessings be upon His last Prophet Muhammad, who encouraged all human beings to seek and share knowledge.

I wish to express my deepest gratitude and affection to my parents: Mrs. Sabiha Hoda and Dr. Najmul Hoda for instilling me with a life-long love for learning; my husband, Mohammed Asif, for being my pillar of strength and best friend; my children, Atif and Imran, for being my biggest source of inspiration; my brothers: Shariq Hoda, for always expecting the very best from me and Dr. Asif Hoda, for nudging me into Computer Science, knowing somehow that it will develop into a life-long passion.

I am extremely indebted to my supervisors, Prof. James Noble and Dr. Stuart Marshall, for seeing me through the ups and downs of this lengthy pursuit, for encouraging me, challenging me, drawing out the best in me, and most of all, for always having faith in me.

I wish to express my warmest gratitude to my friends: Amaara Rehmaan, Mutsumi Tanio, Aneesa Adam, Zeenah Adam, and many others for their unending encouragement and support.

I wish to extend my gratitude to the Agile Professionals Network (APN, Wellington) and Agile Software Community of India (ASCI, India) for providing access to Agile software development communities in which to conduct this research. I thank all my participants for coming forth and sharing their experiences with me. I sincerely hope this research will benefit you.

I thank my fellow Software Engineering researchers and members of the Elvis group at VUW, which include but are not limited to Craig Anslow and Nick Cameron, for their support and encouragement. I am grateful to Prof. John Hine and Dr. Peter Andreae (pondy) for their continued support.

Many thanks to Dr. George Allan, Dr. Angela Martin, Esther Derby, Jennifer Ferriera, Diane Strode, Michael Waterman, and Mary Denton for reviewing various parts of this thesis and providing their valuable recommendations. Thanks to Prof. Philippe Kruchten and Dr. Tore Dybå for their research collaborations and to Dr. Frank Maurer, Shane Hastie, and Steve Adolph for their continued interest in and support of this research.

I am very grateful to BuildIT (NZ) for a PhD scholarship, the Agile Alliance (USA) for an academic grant, Software Process and Product Improvement (NZ), and the School of Engineering and Computer Science (VUW) for their financial support of this research.

Last but not the least, I thank Prof. Helen Sharp (The Open University, UK), Prof. Stephen MacDonell (AUT, New Zealand), and Dr. Petra Malik (VUW, New Zealand) for their time and effort in examining this thesis and for their valuable feedback.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traditional software development teams are composed of individuals with different organizational roles such as developers, testers, designers, business analysts, etc. These roles are well defined, formal roles and the team members function within the boundaries of their separate roles to carry out each of the steps involved in a traditional software development method, such as requirements gathering, analysis, design, implementation, testing, and maintenance. Project managers are responsible for managing the affairs of the team, such as goal setting, task allocation, tracking progress, team evaluations, and improvement. Project managers act as a middle-layer between the team and senior management, conveying senior management expectations to the team and raising any team-wide issues to senior management for resolution. Project managers on traditional teams are also responsible for managing customer relationships and expectations by co-ordinating between the team and their customers.

Agile software development teams, on the other hand, are self-organizing teams [40, 72, 137] composed of "*individuals [that] manage their own workload, shift work among themselves based on need and best fit, and participate in team decision making*" [71]. Self-organizing teams exhibit autonomy, cross-fertilization, and self-transcendence [154] and must have common focus, mutual trust, respect, and the ability to re-organize repeatedly to meet new challenges [40].

1.1 Motivation

Software engineering researchers are exploring the structure and behaviour of Agile software development teams [38, 113, 119, 141], partly in response to the Agile software movement's increasing popularity within industry over the past decade [21, 100]. A majority of these studies have focused on eXtreme Programming (XP) teams [107, 140, 141, 142, 164]. In contrast, research on teams using project-oriented Agile methods such as Scrum (or combinations of Scrum and XP) is extremely limited in comparison to its growing popularity [112]. Recent studies have called for research that is (a) empirical, (b) industry-based, (c) focuses on Scrum, and (d) addresses people and their concerns about adoption of Agile methods [6, 21, 28, 51]. This research addresses all of the above.

The specific topic of this thesis is self-organizing Agile teams—a hallmark of Agile software development and of the Scrum method in particular. Self-organizing teams have been identified as one of the critical success factors of Agile projects [35]. Self-organization can also directly influence team effectiveness [111] as decision making authority is brought to the level of operational problems, which increases the speed and accuracy of problem solving. While Agile software development, and in particular the Scrum method, emphasize self-organizing teams, they do not provide clear guidelines on how self-organization should be implemented [113]. There has been limited research on the subject and almost none across multiple projects, organizations, and cultures. How Agile teams achieve and sustain self-organization in practice is not well understood. This thesis explains how software development teams become self-organizing Agile teams.

1.2 Research Contributions

This thesis presents a grounded theory of self-organizing Agile teams. The grounded theory is based on a research study involving 58 Agile practitioners across 23 different software organizations from the New Zealand and Indian software industries. The resulting theory of self-organizing Agile Teams explains how software development teams take on informal, implicit, transient, and spontaneous *roles* and perform balancing acts on a set of integrated *practices* while facing critical environmental *factors*, in order to become a self-organizing Agile team.

The main contributions of this thesis are:

- **Self-Organizing Agile Team Roles** of *Mentor*, *Co-ordinator*, *Translator*, *Champion*, *Promoter*, *Terminator*, that are informal, implicit, transient, and spontaneously taken up by team members in response to challenges faced by the team.

- **Role of the Agile Coach** in terms of the self-organizing Agile roles they are likely to play during different stages of the team's maturation.

- **Self-Organizing Agile Team Practices** that are performed by Agile teams, while balancing—*freedom and responsibility*, *cross-functionality and specialization*, and *continuous learning and iteration pressure.*

- **Factors Influencing Self-Organizing Agile Teams**: *senior management support* and *level of customer involvement.*

This thesis also presents a discussion of the research results in light of existing literature which further supports the roles and practices identified in this research. A description of Grounded Theory, as a research method, its application in this research with examples, and reflections on the challenges faced in using Grounded Theory and strategies for overcoming them are also presented.

1.3 Thesis Structure

Chapter 1 Introduction Describes the motivations behind this research, the contributions of this research, and the structure of this thesis.

Chapter 2 Literature Review Presents an overview of related literature. In keeping with the research method (described in the chapter 3), a minimal literature review was conducted up front. A detailed literature review is presented in light of the research findings, discussion section, at the end of each of the results chapters 4, 5, and 6.

Chapter 3 Research Design Surveys research perspective and research methods, and then presents a detailed description of Grounded Theory, along with examples of its application, challenges faced in using Grounded Theory and strategies found useful in overcoming them.

Chapter 4 Self-Organizing Agile Team Roles Introduces the theory of self-organizing Agile team roles, practices, and the critical factors that influence them and describes the informal, implicit, spontaneous, and transient self-organizing Agile team roles: *Mentor*, *Co-ordinator*, *Translator*, *Champion*, *Promoter*, *Terminator*. This is followed by a discussion of these roles in light of related literature.

Chapter 5 Self-Organizing Agile Team Practices Describes the practices that enable self-organization in Agile teams: collective decision making, self-assignment, self-monitoring, multiple perspective, group programming, rotation, self-evaluation through retrospectives, and self-improvement through learning spike and pair-in-need. These practices are performed while balancing *freedom and responsibility*, *cross-functionality and specialization*, and *continuous learning and iteration pressure*. This is followed by a discussion of these practices in light of related literature.

Chapter 6 Factors Influencing Self-Organizing Agile Teams Describes the two most critical factors that influence self-organizing Agile teams: *senior management support* and *level of customer involvement*. This is followed by a discussion of these factors in light of related literature.

Chapter 7 Conclusion Describes the *contributions* of this thesis, a *discussion* of related literature, the *limitations* of the study, a discussion of the *implications for practice*, and suggests ideas for *future work*.

Terminology used in this thesis: *Traditional software development* is used as a catch-all phrase for software development methods characterized by a structured software life-cycle with structured phase boundaries, voluminous design and requirements documents, hierarchical organization structures, and manager-led teams. *Agile coaches* refer to XP Coaches and Scrum Masters. *New teams* refers to teams on their first Agile project and/or those with less than an year of experience with Agile methods. *Mature teams* refers to teams with experience of multiple Agile projects and/or those with more than an year of experience with Agile methods. The term '*our*' refers to Rashina Hoda, typically in consultation with her supervisors and is used to differentiate this thesis from other research in the discussion sections 4.8, 5.5, 5.6, 5.7, 6.5, and 7.3.

Chapter 2

Literature Review

This chapter presents an overview of traditional software development models (Waterfall and Spiral) and traditional software development teams. This is followed by a description of Agile software development models (Scrum and eXtreme Programming) and Agile software development teams. Finally, a review of literature on self-organizing teams is presented. Literature related to the research findings is further discussed in detail in discussion sections, at the end of each of the results chapters 4, 5, and 6.

2.1 Traditional Software Development

Several software development models came into use over the years to provide process and structure to the various activities involved in software development. An overview of the Waterfall and the Spiral model, a couple of examples of traditional software development models, is provided below [25, 133].

2.1.1 The Waterfall Model

The Waterfall model was initially proposed by Winston W. Royce in 1970, as a specification-driven approach to software development [132]. Figure 2.1

shows the steps involved in the Waterfall development model such as requirements gathering, analysis, design, coding, testing, and operations.

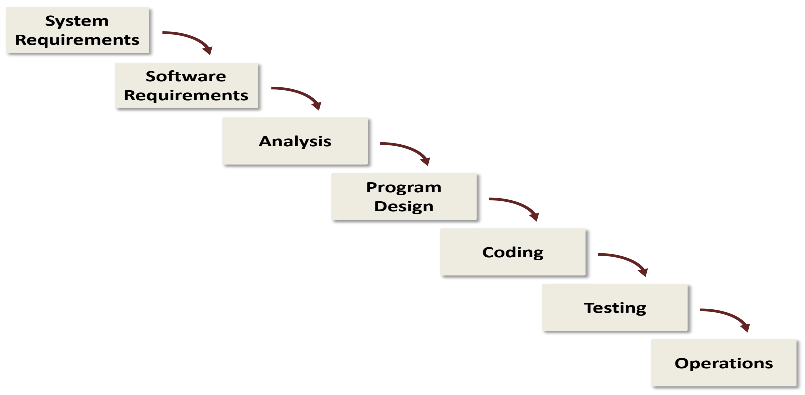


Figure 2.1: The Waterfall Model [132]

The Waterfall model was a modification to the Stagewise or Cascade model documented by Bennington [25, 133] in 1956. The Stagewise model included stages of development: operational plan, operational specification, design and coding specifications, development, testing, deployment, and evaluation [25, 133]. The Waterfall model modified the Stagewise model by including a feedback loop to allow previous stages to be revisited [133]. The Waterfall model was intended to be somewhat iterative in nature ("*build it twice*"), however its purely sequential form evolved into the popular notion of Waterfall [93, 155]. In the sequential form of the Waterfall model, all the requirements were gathered up-front before commencing any design, all the design was completed for the entire project before starting any development, and so on [25].

The classic weakness associated with the Waterfall method is poor flexibility [27, 129, 119]. In real life development, it is common to experience frequent changes in customer requirements. The Waterfall model unrealistically assumes that the customer requirements can be gathered all at once at the beginning of the project and that they remain largely unchanged over the entire length of the project. This leaves little scope for accommodating changes in requirements later in the project. Another weakness of traditional methods is that the Big Design Up Front (BDUF) is an expensive exercise [119, 144]. The amount of time and effort spent in planning and designing a solution may all go to waste in the face of frequently changing project requirements.

2.1.2 The Spiral Model

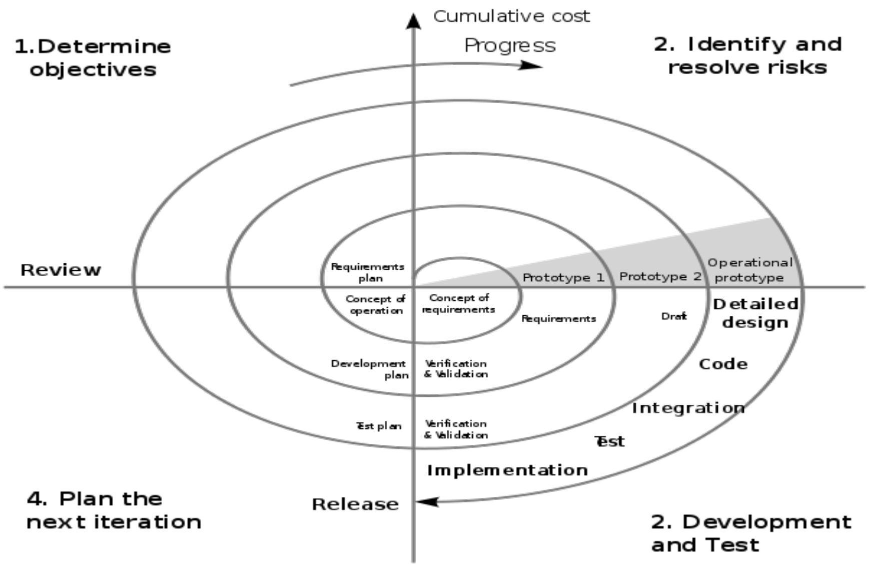


Figure 2.2: The Spiral Model [163]

The Spiral model of software development was introduced by Barry Boehm [25] in 1986, as a risk-driven approach to software development[133]. Figure 2.2 shows an overview of the Spiral model, where each iteration goes through the phases of determining objectives, evaluating alternatives and identifying and resolving risks, development and testing, and planning the next iteration. The Spiral model involves identifying and analyzing risks, both performance-related and development-related. Each cycle involves building a prototype with minimum risks, which is verified and validated. The primary strength of the Spiral model is that it identifies any major risks associated with the project quickly. An advantage of the Spiral model over the Waterfall model is that the Spiral model allows the customers or users to preview the prototypes. The primary weakness of the Spiral model is that the amount of time and effort spent in identifying risks provides little returns for low-risk projects. Another weakness of the Spiral model is its reliance on the system designers' to correctly identify risks for the upcoming cycle and the unrealistic assumption that designers can foresee all problems without actual implementation [25, 133].

2.1.3 Traditional Software Development Teams

Traditional software development is characterized by manager-led teams, organized in a hierarchical structure with multiple layers of authority [158]. Management in traditional teams is typically command and control style [119]. Roles on traditional teams are based around functional tasks reflected by their organizational roles, such as programmers responsible for programming, testers responsible for testing, analysts responsible for requirements analysis, etc. Work is delegated to team members by their managers. Practices of traditional teams include documentation, specifications, and planning [118, 119]. 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 [158].

The Chief Programmer team and the Surgical team are examples of hierarchical teams designed to tackle large software systems development [30, 109]. 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 [30]. 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, non-product 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 [158].

2.2 Agile Software Development

Agile software development methods emerged in the late 1990s [94]. Agile methods follow an iterative and incremental style of development where collaborative self-organizing teams dynamically adjust to changing customer requirements [85, 100, 108]. The developers of some of these methods collaboratively wrote the Agile Manifesto [72] and use '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 perceived weaknesses of traditional software development models [129]. Agile methods are meant to improve over the traditional software development models by accommodating changes through 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 flavours of Agile methods include: *Dynamic Software Development Method* (DSDM), referred to as the first Agile method [5, 51, 93, 146]; *Crystal*, a family of methodologies consisting of a number of methods, and principles for customizing them for particular projects [5, 39]; *Feature Driven Development* (FDD), which focuses on features-based division of work [124]; and *Adaptive Software Development* (ASD), which focuses on concepts and culture, and creating emergent order "*out of chaos*" [5, 70].

Scrum and *XP* are the most widely adopted Agile methods in the world [127]. Most XP practices are focused around development activities at the team level: in contrast, Scrum focuses more on project management [6, 51]. A detailed description of Scrum [43, 139] and XP [19] based on literature in terms of their team roles, practices, artifacts, and ceremonies, is provided below.

2.2.1 Scrum

Scrum was developed by Jeff Sutherland and formalized by Ken Schwaber [139]. Scrum derives its roots from Takeuchi and Nonaka's paper in 1986 "*The New New Product Development Game*" in the Harvard Business Review [154].

Scrum is characterized by **Sprints** work cycles typically 2 to 4 weeks [43]. 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. Figure 2.3 shows a typical Scrum sprint. Table 2.1 provides a description of the main roles, artifacts and ceremonies in Scrum. A description of basic Scrum roles, artifacts, and ceremonies based on literature [43, 139] is provided below.

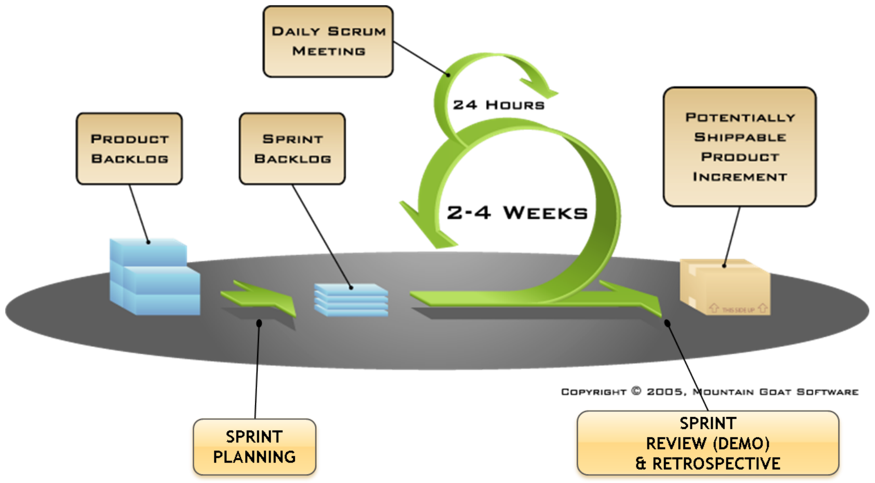

Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
Copyright © 2005, Mountain Goat Software
Sprint Planning
Sprint Review (Demo) & Retrospective

Figure 2.3: A Typical Scrum Iteration [145]

Scrum Artifacts

- **Product Backlog** is the list of features prioritized by business value delivered to the customer. The Product Backlog includes all the features visible to the customer as well as technical requirements needed to build the product.
- **Sprint Backlog** is a subset of the Product Backlog and contains the prioritized features to be developed in a given sprint.
- **Burndown Chart** displays the cumulative work remaining on a daily basis and helps guide the development team towards an on-time and successful sprint.

Scrum Roles

- **Product Owner** is a customer representative, responsible for the ultimate purpose of the product, a business plan, and a road-map that

Table 2.1: Scrum Roles, Artifacts, and Ceremonies [139]

Roles	
Team	A cross-functional team typically of seven plus/minus two members, responsible for selecting the sprint goal and organizing themselves to achieve them.
Scrum Master	A facilitator, responsible for ensuring the team is fully functional and productive, removing impediments, protecting the team from external interferences, and ensuring that the process is followed.
Product Owner	A customer representative, responsible for defining and prioritizing the product features and providing feedback to the team.
Artifacts	
Product Backlog	A list of features prioritized by business value, provided by the customer.
Sprint Backlog	A subset of the Product Backlog and contains the prioritized features to be developed in a given sprint
Burndown Chart	A graph displaying the cumulative work remaining on a daily basis, designed to guide the development team towards an on-time and successful sprint.
Ceremonies	
Daily Scrum	A fifteen minute meeting designed to allow team members to report status.
Sprint planning meeting	A meeting where the team and their customer representative discuss the Product Backlog, and develop a detailed plan for the next sprint.
Demo	A demonstration of the working software developed by the team in a sprint, to the Product Owner.
Retrospective	A meeting where the team members collaboratively discuss their performance in the previous sprint, and identify strategies for improvement.

chalks out multiple releases. The Product Owner prepares the Product Backlog with help from the team. The Product Owner is responsible for: defining the features of the product, deciding release dates and the profitability of the product, prioritizing product features according to market value, adjusting features and priority every 30 days as needed, and accepting or rejecting work results.

- **Scrum Master** is a facilitator that works closely with the team and the Product Owner. The Scrum Master should be aware of the tasks that have been completed, new tasks that have been identified, and any estimate changes. They are responsible for noting and removing impediments faced by the team. They also help resolve any differences or issues amongst team members to ensure full productivity. The responsibilities of the Scrum Master include: ensuring the team is fully functional and productive, enabling close co-operation across all roles and functions, removing impediments, protecting the team from external interferences, and ensuring that the process is followed.

- **Team** is cross functional and has typically seven plus/minus two members. The team selects the sprint goal and specifies work results. The team has the right to do everything within the boundaries of the project guidelines to reach the sprint goal. The team organizes itself and its work, and demonstrates work results to the Product Owner.

Scrum Ceremonies

- **Daily Scrum** also known as a daily standup, is a fifteen minute meeting designed to report the status of the sprint. The Scrum Master leads the team every day in their daily standup meeting, where team member answers three questions: *What did I do yesterday? What will I do today? What impediments are in my way?*

- **Sprint planning meeting** is a meeting where a detailed plan for the sprint is developed. In the sprint planning meeting, the Product Owner reviews the road-map, vision, release plan, and Product Backlog with the team. The team sets and reviews the estimates for the features. Updates on the sprint are provided by the Scrum Master and goals for the next sprint are set.

- **Demo** or demonstration is a session where the team demonstrates the features developed in a given sprint to the Product Owner. A demo can be held during the first half of the sprint planning meeting or in a separate session.

- **Retrospective** is a meeting where the Scrum Master leads the team into collaboratively identifying positive ways of working and strategies for improvement.

2.2.2 eXtreme Programming (XP)

eXtreme Programming (XP) was developed by Kent Beck, with support from Ward Cunningham, Ron Jeffries, and Martin Fowler [19]. 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*" [19]. 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 value. 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. A description of basic XP roles, values, and practices based on literature [19] is provided below.

XP Roles

- **Coach** is responsible for the process as a whole. The Coach needs to remain calm in stressful situations and guide the team. The coach also needs to understand the process and learn from other XP teams.
- **Tracker** is responsible for making good estimates and checking how they match up to the real results. With practice and feedback, the tracker should be able to make good calls on the status of the iterations and releases: whether the team is on the schedule and if any major changes are in store. They need to be able to collect information without disturbing the entire process. The tracker has been called the conscience of the team by Beck [19].
- **Programmer** should possess good communication skills and maintain simplicity in work and code. The programmer is called the heart of XP.
- **Customer** is meant to be an integral part of the XP team. They need to learn how to write stories, to write functional tests, to make decisions, and to demonstrate courage.
- **Tester** helps the customer to write functional tests, runs them regularly, and posts results for everybody's knowledge.
- **Consultant** may be needed to assist the XP team. The job of the consultant is to provide technical knowledge or to help with the process.
- **Big Boss** is responsible for the project and is the project sponsor. The Big Boss needs to check the team's progress regularly and should practice honest communication with the team. The Big Boss should take time to listen to the teams' issues.

Table 2.2: XP Roles and Values [19]

Roles	
Coach	A person responsible for the process as a whole, guiding the team, and understanding the process and learning from other XP teams.
Tracker	A person responsible for making good estimates and checking how they match up to the real results.
Programmer	A person with good communication skills, who maintains simplicity in work and code.
Customer	A person(s) from the customer organization, who is integral part of the XP team.
Tester	A person who helps the customer write functional tests, runs them regularly, and posts results for everybody's knowledge.
Consultant	A person who may be needed to assist the XP team by providing technical knowledge or help with the process.
Big Boss	A project sponsor, responsible for checking the team's progress regularly and communicating with the team to listen their issues.
Values	
Communication	XP practices such as unit testing, pair programming, and task estimation encourage communication channels to remain open at all times.
Simplicity	Implies that the team concentrates on something simple today which may require changing tomorrow, rather than spend too much time and effort on something complicated that may never be used later.
Feedback	The programmers get feedback about the state of their system through unit tests; the customers receive feedback from programmers through estimation of user stories; the customer provides feedback to team through reviews.
Courage	It requires courage to address issues, fix the problems, and throw away code in favour of alternative better designs and implementations.
Respect	XP requires honest communication and close collaboration between all members of the team.

XP Values

- **Communication:** The lack of sufficient communication between people can lead to serious problems in a project. XP advocates communication between programmes, customers, and managers. XP practices such as unit testing, pair programming, and task estimation are aimed at encouraging communication channels to remain open at all times.

- **Simplicity:** The team responds to the question *What is the simplest thing that could possibly work?* Simplicity implies that the team concentrates on something simple today which may require changing tomorrow, rather than spend too much time and effort on something complicated that may never be used later.

- **Feedback:** By writing unit tests for the system, the programmers are meant to get feedback about the state of their system. The customers are supposed to receive feedback from programmers in the form of estimation of new user stories (description of features). The customers review the schedule to provide feedback about the team's velocity. Concrete feedback is meant to encourage communication.

- **Courage:** It requires courage to address issues in the middle of development and fix the problems while maintaining project velocity. It also takes courage to throw away code in favour of alternative better designs and implementations. Communication promotes courage by allowing experimentation, which in turn supports simplicity as the team is always encouraged to try to simplify the system.

- **Respect:** Beck added this fifth value in the second edition of his book [20]. XP requires honest communication and close collaboration between all members of the team. This is not possible without high levels of trust and respect between programmers, managers, and customers.

XP Practices There are twelve XP practices [19]. Figure 2.4 shows the different XP practices and the arrows between them show how they relate to, and support each other.

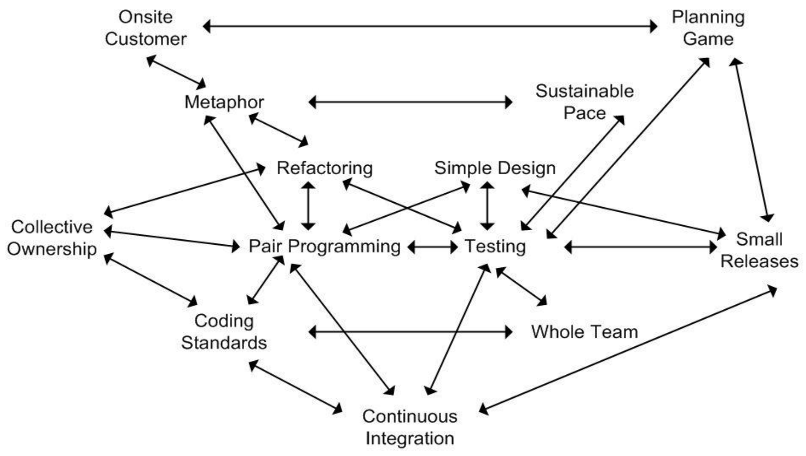


Figure 2.4: XP Practices [19]

- **Planning Game** is a meeting where projects are planned. The Planning Game involves the business taking decisions about scope, priority, composition and dates of releases, and the technical people taking decisions about estimates, consequences, process, and detailed scheduling. The next iteration is planned based on the features prioritized by the customer and the work estimated by the programmers.

- **Small Releases** allow features to be developed quickly in short cycles.

- **Metaphor** or a simple shared story guides system development and communication.

- **Simple Design** advocates choosing the simplest design possible and removing any unnecessary complexity as soon as it is discovered.

- **Testing** involves unit tests written by programmers that guide the code, and acceptance tests that define whether an implementation is complete.

- **Refactoring** involves restructuring the system to simplify, remove duplication, improve communication, or add flexibility, without changing the behaviour of the system.

- **Pair Programming** is the practice of two programmers working together on one workstation and collaborating on all aspects of the programming. One partner, termed the *driver*, works with the keyboard and mouse, while the other, termed the *navigator*, maintains a more strategic view.

- **Collective Ownership** implies all code is owned by everyone and can be changed at anytime to the advantage of the system and design. As pairing of programmers is dynamic, everyone has the opportunity to learn something about every part of the code.

- **Continuous Integration** involves integrating and building the system several times a day, after each task is completed.

- **40-hour week** ensures programmers are fresh and eager every day. The rule also dictates that no one can work a second week of overtime.

- **On-site customer** is the business representative, who is available to set priorities, answer questions that programmers may have.

- **Coding standards** imply that programmers will endeavour to write code in accordance with rules that focus on communication and maintain uniform set of coding practices.

XP is different from traditional methodologies in the following ways [19]: short cycles, early and continuing feedback, and an incremental approach; implementation of functionality to be flexibly scheduled while responding to changing business needs; reliance on oral communication, automated tests, and source code to describe system structure and intent.

2.2.3 Agile Software Development Teams

A hallmark of Agile software development is its focus on people and social interactions. The values of the Agile Manifesto promote a people-focused view of software development. It is no surprise, therefore, that researchers are now exploring the structure and behaviour of Agile software development teams [38, 104, 108, 119, 141, 162, 164], in response to the Agile software movement's increasing popularity within industry over the past decade [21, 100, 119]. A systematic review of empirical studies of Agile software development found that about 20% of research studies on Agile software development focused on human and social factors [51].

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 [119]. Management in Agile teams is meant to be facilitative and co-ordinating [119]. Smaller teams are better suited to democratic structures than larger teams [158]. This is one of the reasons that Agile teams work best in smaller numbers [155, 119].

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 [119]. 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. The 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 manager to relinquish their authority [119].

A popular slogan "*people trump process*" highlights the importance of people in Agile software development [40]. Cockburn et al. 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 [40]. They argue that Agile organizations practice "*leadership-collaboration*" instead of command and control style management, and that management in Agile organizations trust 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 et al. have conducted an extensive ethnographic study of five mature XP teams, describing characteristics of XP teams [141], collaboration and co-ordination in XP teams [142], the effect of different organizational cultures on the practice of XP [130], and the social aspects of XP's technical practices [131]. Their study confirms the highly collaborative and self-organizing nature of Agile teams [142]. Sharp et al. describe the culture of mature XP teams as possessing five characteristics: (a) respect on both an individual and team level, (b) responsibility on both an individual and team level, (c) maintaining quality of working life, (d) confidence in their own abilities coupled with constant re-validation and re-affirmation, and (e) trust, that underpins the other four. Their study established the importance of story cards (physical cards that contain the description of a user story) and story walls/boards (physical walls/boards that comprises of the user stories that the team has committed to implementing in a given iteration, along with their break-down into technical tasks) in collaboration and co-ordination within XP teams [141]. While simple, these physical artifacts proved to be information rich focal points for collaboration and co-ordination.

Williams et al. have extensively researched XP's pair programming practice [164, 165]. Pair programming has been shown to improve productivity and quality of products [165]. 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. The 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 literature [106, 105]. Martin's study describes an informal XP customer team that consist of different roles, where the Negotiator was the closest to the on-site customer defined in literature. The 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 [162]. The findings 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. This study calls for more studies to be conducted on social and cultural issues on Agile teams, specially with regards to "*self-regulatory*" work structures [162].

Most of the above research has focused almost exclusively on XP teams [107, 140, 141, 142, 164]. In contrast, research on Scrum is scarce, despite Scrum being arguably the most popular Agile method used in the industry [51, 113].

2.3 Self-Organizing Teams

The concept of self-organizing teams existed long before it was formally incorporated as a hallmark of Agile software development [72]. This section presents a review of self-organizing teams from several perspectives: socio-technical systems perspective, organizational theory perspective, complex-adaptive systems perspective, knowledge management perspective, and finally, an Agile software development perspective.

2.3.1 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 [159]. 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 [17]. Concertive control was argued to be an alternative to the bureaucratic control marked by an hierarchical system with rational-legal rules rewarding compliance [97]. 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 [17].

Self-managing teams were described as teams made up of 10 to 15 people taking on the responsibilities of their former supervisors; whose every day 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 co-ordinated with other areas of the company [17]. 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 team proposed in Agile software development is inescapable [71, 136].

2.3.2 Organizational Theory Perspective

Self-organizing teams have been described from an organizational theory perspective [86, 114, 115]. 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 [1, 115]. Four principles of self-organization in a holographic organization are defined as: minimum critical specification, requisite variety, redundancy of functions, and learning to learn [14, 115]:

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 [115]. Morgan also emphasizes the need for self-organizing teams to work in an environment of "*bounded*" or "*responsible autonomy*" [115]. The role of management is extremely important in providing autonomy to the team and for team empowerment [86].

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 [115]. In other words, the organization must match the variability of its external environment. Requisite variety implies that changes in the environment of the organization is 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 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 influential in improving the quality of the employee's working life [86, 114].

The principles of requisite variety and redundancy of functions are closely related. Redundancy of functions, refers to the multi-functionality of workers where workers are able to perform a wide variety of team tasks through cross-training [86].

Learning to Learn refers to the team's ability to reanalyze problems, reappraise the best working method, and reconsider the required output if necessary [86]. Sustenance of self-organization requires double-loop learning, where the rules and norms adapt to changing environments [1].

The holographic organizations metaphor has been theoretically explored in the context of self-organizing Agile teams by Nerur et al. [118]. Minimum project planning and specification up-front 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, standup 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.

2.3.3 Complex Adaptive Systems Perspective

Self-organization has also been discussed from the complex adaptive systems perspective [16, 88, 92, 96, 99, 154]. Complex adaptive systems (CAS) are systems that exhibit spontaneous order through a process of self-organization [92]. Immune systems, ant colonies, human cities, and eco-systems are examples of complex adaptive systems [92]. Kauffman explored CAS in human organizations and economics, defining modern organizations as self-sustaining structure of roles and obligations [88]. Levin further suggested that co-operation and networks of interaction emerge out of individual behaviours and in turn influence them [96].

Anderson et al. [10] define self-organizing teams as teams that are (a) informal and temporary, (b) formed spontaneously around issues (c) are not a part of a formal organization structure, (d) have a strong sense of shared purpose, (e) where team members decide their own affairs, and (f) 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 [16]. 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 [98, 99].

2.3.4 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 to possess self-organizing capability when it exhibits three conditions: autonomy, cross-fertilization, and self-transcendence [154]. A team exhibits autonomy when they are provided freedom by their senior management to manage and assumes responsibility of their own tasks and when there is minimum interference from senior management in the team's day to day activities [154]. A team exhibits cross-fertilization when it is composed of individual members with varying specializations, thought processes, and behaviour patterns and these individuals interact amongst themselves leading to better understanding of each others perspectives [154]. 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 [120]. Self-organizing teams accumulate and spread knowledge through (a) "*multilearning*" made up of multilevel learning across individual, group, and organizational levels and "*multifunctional learning*" across functions, and (b) "*transfer of learning*" across different departments of the organization [154]. The self-organizing team with its cross-functional and multiple learning capabilities replaced traditional teams with specialists in particular knowledge areas.

2.3.5 Agile Software Development Perspective

Finally, from an Agile software development perspective, self-organizing teams are at the heart of Agile software development [35, 40, 72, 108, 137, 141]. Self-organizing teams are considered the source of best architecture, requirements, and design [72]. While Scrum specifically mentions self-organizing Agile teams, the concept of "*empowered*" teams has only recently been added to XP [166].

Self-organization is one of the principles behind the Agile Manifesto and has been identified as one of the critical success factors of Agile projects [16, 35, 72]. 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*" [71]. Self-organizing teams must have common focus, mutual trust, respect, and the ability to organize repeatedly to meet new challenges [40].

Sutherland, a co-creator of Scrum, explains that self-organizing teams consist of "*members with diverse backgrounds*" who are "*given a free hand*" by the top management [152]. Schwaber, the other co-creator 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*" [136]. Schwaber suggests individuals on the team need to co-ordinate 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 [95]. 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 [24, 54].

Self-organizing Agile teams are not leaderless, uncontrolled teams [40, 154]. Leadership in self-organizing teams is meant to be light-touch and adaptive, providing feedback and subtle direction [11, 16, 34, 154]. Leaders of Agile teams are responsible for setting direction, aligning people, obtaining resources, and motivating the teams [11].

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 [113]. 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 [112]. The term *self-managing*, in that paper, is used to describe Agile teams and is considered synonymous to *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, co-ordination, and communication [50]. The results show that the main challenges to achieving team effectiveness include problems with team orientation, leadership, and co-ordination, 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 practitioner-based literature on self-organizing Agile teams abound, research literature on the subject is scarce. Some studies on Agile teams have acknowledged the self-organizing nature of Agile teams [141, 162]. 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 [113]. Moe et al. note that Agile methods, specially Scrum, emphasizes self-organizing teams but do not provide clear guidelines on how they should be implemented [112, 113]. 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 thesis presents 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.

Most of the literature pertaining to self-organizing teams presented here is revisited, and further literature is discussed in relation to the research findings as discussion sections 4.8, 5.5, 5.6, 5.7, and 6.5.

Chapter 3

Research Design

Software engineering researchers are constantly looking to improve the quantity and quality of their research findings through the use of an appropriate research method [143]. Over the last decade, there has been a sustained increase in the number of researchers exploring the human and social aspects of software engineering through qualitative research methods [32, 45, 41, 107, 162]. This chapter provides a description of our choice of research method and research perspective, role of the researcher, and the theory and application of Grounded Theory in this research.

3.1 Research Methods

This section provides a brief description of different research methods considered, and presents our motivation for choosing Grounded Theory. Creswell [46] and Oates [122] present detailed descriptions of various research methods and designs.

Survey Research: Survey research allows capturing data from a broad population with the aim of identifying their characteristics [52, 122, 143]. Survey research often makes use of questionnaires to collect data from a large number of individuals. Formulation of a clear research question and careful selection of a representative subset of the population are prerequisites for Survey research. This research was driven by a motivation to explore the human and social aspects of Agile teams. Since there was no clear research question or hypothesis to begin with, survey research was not a suitable option for conducting this research.

Case Studies: Case Studies, used as a research method, enables the study of a contemporary phenomenon in its natural setting, specially "*when the boundaries between phenomenon and context are not clearly evident*" [116, 167]. Case Study research can be single-case or multiple-case. The cases are selected based on their relevance to a pre-formulated research question [52].

Ethnography: Ethnography finds its roots in Anthropology. The aim of Ethnography is to study community of people in order to understand how they make sense of their social interactions [52, 141]. Researchers using Ethnography often become a member of the community for the duration of the observations. Ethnographies often result in rich descriptions of the community that help define its culture [141]. Ethnography is well suited to explore the social aspects of Agile teams [131, 141, 142].

Grounded Theory: Grounded Theory, used as a qualitative research method, studies people and interactions in order to capture the main concern of the participants and how they go about resolving it. A detailed description of the Grounded Theory method and its application in this research, is provided in the rest of this chapter.

Grounded Theory was selected as the method for this research. Strong institutional support and a successful history of using Grounded Theory for exploring human and social aspects of Agile teams [107, 106] within the department were the primary reasons for selecting Grounded Theory over other applicable methods, such as Ethnography. Other reasons include the following: firstly, Agile methods focus on people and interactions and Grounded Theory, used as a qualitative research method, allows the study of social interactions and behaviour [126]. Secondly, Grounded Theory focuses on theory generation, rather than extending or verifying existing theories—an interesting and exciting prospect. Thirdly, Grounded Theory is useful when studying relatively new areas or when trying to gain a fresh perspective on a well-known area [147] and there has been limited research on the human and social aspects of Agile software development. Finally, Grounded Theory has been used successfully, and continues to gain popularity, as a research method to study Agile software development teams around the world [38, 41, 107, 162].

3.2 Research Perspectives

Research can be carried out using different underlying philosophical perspectives, such as: *positivist*, *interpretive*, and *critical* [36, 52, 116, 123].

Positivist: A positivist view of the world assumes that knowledge is based on inferences from observable facts [52]. Positivists assert the study of a phenomenon is independent of the researcher and their tools [116]. The main focus of a positivist perspective is to test theory in order to "*increase predictive understanding of phenomena*" [116, 123]. Examples of research methods most commonly associated with a positivist approach are Survey Research and Case Studies [52, 116], although Case Study research is also used with other research perspectives.

Critical: A critical perspective assumes that "*research is a political act*" [52]. Researchers following the critical approach are referred to as critical theorists. The main focus of the critical theorists is to study conflicts in society and take on an emancipatory role [52, 116]. The research method most commonly associated with a critical approach is Action Research [52].

Interpretive: An interpretive perspective endorses the idea that scientific knowledge is inherently inseparable from its human context and that reality can be studied through social constructs, such as language [52, 116]. The main focus of an interpretive perspective is to study a phenomenon by understanding how people make sense of it. An interpretive perspective rejects an objective view of the world and does not attempt to generalize from sample to population. As a result, the findings derived using this perspective are closely tied to the context of the study. An interpretive perspective leads to a deep understanding of the phenomenon in a sample context which can then be used to inform other contexts [123].

In this research, Grounded Theory was used with an interpretive perspective since (a) the focus was to generate theory, rather than verify existing ones (which rules out a positivist perspective), and (b) the conceptual findings resulting from the study, although modifiable, are grounded in the contexts studied. [116].

3.3 Role of the Researcher

Since this Grounded Theory research was carried out using an interpretive approach, the role of the researcher is important in how the phenomenon under study is interpreted. This section provides a background of the researcher.

I completed a Bachelor of Science with honours distinction in Computer Science from Louisiana State University, USA in 2003. My personal interest in literature guided me into taking several elective courses in English literature. One of these courses—based on critical thinking and writing—particularly helped me view a phenomenon with an open mind and express it from multiple and distinct perspectives.

Thereafter, I worked in the Indian software industry for one and a half years, at Ebookers plc (a web-based, pan-European travel agency). As a developer, I was exposed to the inner workings of software development teams, their management, and customers in a traditional setting. Towards the end of my job, there was a marked move towards more Agile-like projects.

In 2005, I joined the Masters program at Victoria University of Wellington, New Zealand. It was in my first year of my Masters degree that I was introduced to Agile software development, taught as a part of a university course by Dr. Stuart Marshall. I got further interested in the area as a part of an object-oriented paradigms course, taught by Prof. James Noble. Based on my strong academic record and research potential I was admitted to a direct PhD program, under the supervision of Prof. Noble and Dr. Marshall in the area of Agile project management.

Since 2006, I have conducted this research as part of my doctoral degree in New Zealand and India. Being a newcomer to the New Zealand culture, I had no preconceived notion of how the New Zealand software development industry worked. Being an Indian by descent and having worked for a brief period in the Indian software industry, however, meant I had a reasonably good understanding of software development practices in India. This experience worked to my advantage in accessing organizations for participation in research. At the same time, I was conscious not to let this experience cloud the research as I carefully approached interviews and observations with an open mind.

In order to preserve consistency in the application of the research method, I have personally conducted all data collection through interviews and observations, and all the data analysis, with frequent feedback from my supervisors, colleagues, peers, and industry practitioners.

3.4 Grounded Theory

Grounded Theory (GT) 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*" [59]. GT was developed by Barney Glaser and Anselm Strauss, as a result of their collaborative research

Table 3.1: Grounded Theory Terms and Descriptions [81]

Term	**Description**
Minor Literature Review	The researcher can start off with a light literature review—enough to carry on a conversation with the participants.
Theoretical Sampling	A process which allows the researcher to collect, code, and analyze the data and then decide what data to collect next [58]
Open Coding	The first step of analysis and starts by collating key points from raw data. These are then assigned a code—a phrase that summaries the key point in 2 or 3 words [57].
Constant Comparison Method	A process by which codes arising out of each interview are constantly compared against the codes from the same interview, and those from other interviews and observations, producing higher levels of data abstraction [57, 58].
Memoing	The ongoing process of writing theoretical notes throughout the GT process. Memos capture the conceptual links between categories as the researcher notes down their reflections on different categories.
Core Category	Several categories emerge as a result of data analysis and the one that is able to account for most variations in the data and relates meaningfully and easily with other categories is called the core category [58].
Selective Coding	Once the core category is established, the researcher ceases open coding and uses selective coding—a procedure where they code for only the core category and those categories that are closely related to the core.
Theoretical Saturation	When further data collection and analysis on a particular category leads to a point of diminishing results—no new insight into the category is generated—the category is said to have reached *Theoretical Saturation* [58]. The researcher can then stop collecting data and coding for that category.
Major Literature Review	As the theory starts to emerge, the researcher can conduct extensive literature review to see how the literature in the field relates to their emerging theory.
Sorting	Once the researcher has nearly finished data collection and coding is almost saturated, they can begin arranging the theoretical memos on a conceptual level or *Sorting.* Sorting results in an outline of the theory describing how the different categories relate to the core category [58].
Theoretical Coding	Glaser lists several common structures of theories or theoretical coding families [63] which can be used as a framework to describe how the categories relate to each other as a hypotheses to be integrated into a theory. This is called *Theoretical Coding.*
Write up	The final step in GT is writing up the theory, which follows the theoretical outline generated as a result of sorting and theoretical coding.

on dying hospital patients [65]. They published their book *The Discovery of Grounded Theory* (1967) which laid the foundations of GT [65].

The goal of GT is "*to generate a theory that accounts for a pattern of behaviour which is relevant and problematic for those involved*" [58]. 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 [59]. The nature of the 'theory' generated by the Grounded Theory method is best understood as an explication of the research findings [8]. It has also been described as "*a general pattern of understanding*" [46]. In generating a theory, a GT researcher uncovers the main concern of the research participants and how they go about resolving it. The distinguishing features of the GT method are a rigorous analysis method powered by constant comparison of data, called *Constant Comparison method*, and the practice of frequently recording reflections on data in order to elicit relationships between them, called *Memoing* (described later in this chapter.)

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 [33, 62]. 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 [64].

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. Table 3.1 provides a glossary of general GT terms [79]. Figure 3.1 presents an overview of the GT method or *the GT life-cyle* [81]. The diagram captures the main procedures of the Grounded Theory method but does not imply a linear sequence because GT procedures are "*cycled and go on simultaneously, sequentially, subsequently, serendipitously*" [58].

The following sections describe the GT procedures in the order presented in Figure 3.1. Challenges faced in applying the various procedures of the GT method in software engineering research and the strategies found useful in

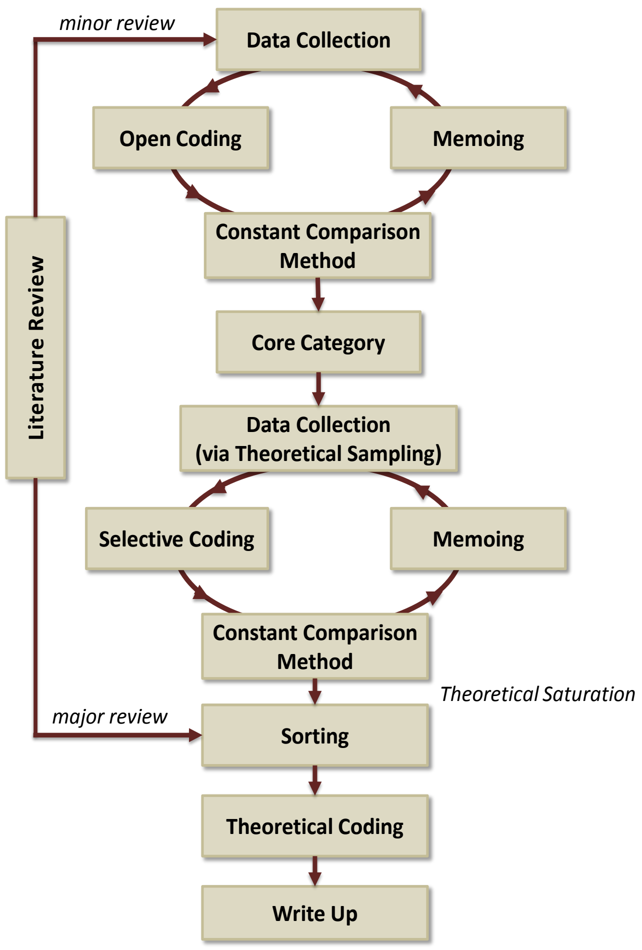


Figure 3.1: The Grounded Theory Life-Cycle [81]

overcoming them are also discussed.

3.4.1 Research Area

In order to effectively study and uncover the main problems of the participants, GT recommends refraining from formulating a research problem or a question up front [58]. The rationale behind this recommendation is that (a) the GT method is meant to generate new theory, and having a preconceived research problem can cause the researcher to be limited in their explorations;

and (b) the research problem should be the problem of the participants under study and should not be preconceived or forced, rather it should be allowed to emerge [58].

Although the researcher is advised against formulating a research question up front, they are required to choose a general area of interest. The plethora of subject areas within software engineering makes choosing one a daunting task. 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 [38, 107, 113, 119, 141, 162].

3.4.2 Minor Literature Review

Glaser's stance on literature review in the GT method has been a topic of debate [149, 156]. While GT does not involve formulating a hypothesis up front based on extensive literature review, the use of literature is not prohibited in the GT method. Glaser strictly warns against extensive literature review in the *same area* of research during the *early stages* of the GT method [58]. Glaser insists that "*undertaking an extensive literature review before the emergence of the core category violates the basic premise of GT*" [62]. The rationale behind a minimal literature review before the emergence of the core category is in many ways the same as that behind not starting with a specific research question, namely: avoid clouding the researcher's mind with preconceived ideas and focusing on generating theory rather than verifying existing theories [58].

Following Glaser's advice, literature review was kept to a minimum 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.

While extensive literature review in the same substantive area as the research is discouraged early on, 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 [58]. Reading articles and dissertations describing research conducted using GT in other areas, for example [23, 53, 91, 117], was found to be useful.

3.4.3 Data Collection

This section describes how the participants were recruited and interviews and observations were conducted. Data collection in GT is guided by a process called *Theoretical Sampling*, which is an ongoing process which helps decide what data to collect next based on the emerging theory:

> "*Theoretical sampling is the process of data collection for generating theory whereby the analyst jointly collects, codes, and analyzes his data and decides what data to collect next and where to find them, in order to develop his theory as it emerges.*" [58]

Recruiting Participants

The search for participants commenced once Human Ethics Committee (HEC) approval was received (Appendix B). Finding participants can be difficult at best and extremely challenging at worst. In the early period of this research there was no umbrella organization or user group for Agile practitioners in New Zealand. Individual Agile companies and practitioners were contacted, with limited success. At an event organized by some Agile companies in New Zealand, the opportunity to meet and interact with several Agile practitioners presented itself. Some of these practitioners offered to participate in our research. The foundations of an umbrella Agile group, the Agile Professionals Network [12] were laid at this very event. However, it was some time before the group grew and became active. The struggle to find research participants continued in the interim and other destinations for data collection were explored. 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 [4, 150, 151, 153, 158, 161]. In exploring the Indian software industry resources online, the Agile Software Community of India, was discovered [13]. A request for participation was emailed to ASCI's user group mailing list, and fortunately, several practitioners came forth to help.

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 of more mature teams towards later stages of the research. A need to include participants from different functional 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, developer, tester, business analyst, designer, customer representative, and senior management. Data collection by theoretical sampling helped develop the emerging theory by (a) adapting questions to focus on emerging concerns (b) choosing participants that were well placed to provide information on the emerging concerns.

This research is based on 58 participants from 23 different software organizations. Of these, 26 were from 10 New Zealand organizations, 28 were from 9 Indian organizations, and 4 were from 4 organizations in North America. Interviews with Agile practitioners in New Zealand were conducted in Wellington. Interviews with 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 Agile2008 conference in Toronto. The domains included health, social services, telecom, entertainment, agriculture, oil and energy, Agile software development and consultancy, etc. The products and services offered by the participants' organizations included web-based applications, front and

Table 3.2: Participants and Projects (P#: Participant Number, Position: Agile Coach (AC), Agile Trainer (AT), Developer (Dev), Customer Rep (Cust Rep), Business Analyst (BA), Senior Management (SM), Knowledge Strategist (KS); *Organizational Size: XS<50, S<500, M<5000, 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

back-end functionality, and local and off-shored software development services. The projects' durations varied from 2 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 3.2 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 standup), 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 5 years of experience on Agile projects. Half of the participants were collaborating directly and regularly with their customers. The other half of the participants were suffering from inadequate customer involvement of some kind—due to either quantity or quality of customer involvement. Over the course of the study (2006—2010), however, there was a marked improvement in the level of awareness and popularity of Agile methods and consequently, in the level of customer involvement. In order to respect their confidentiality, the participants are referred to by numbers P1 to P58.

Interviews and Observations

Data was collected through interviews and was supplemented by observations, over a period of 3 years. Face-to-face, semi-structured 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]?*

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

- *Do you believe that your team is self-organizing? If yes, why? what makes you self-organizing?*
- *What has been the level of customer involvement on this project?*

In addition to interviews, observations were made about the participants' workplaces, such as seating and set-up of information radiators, and several Agile practices, such as daily stand-up meetings (co-located and distributed), release planning, iteration planning, and demonstrations. Observations were made for two teams in New Zealand and three in India for approximately four hours each. Observations help provide greater insight into the data provided through interviews as well as help validate the authenticity of interview data.

Figure 3.2 shows the physical setup of a NZ team. A sample field note from an observation is given below:

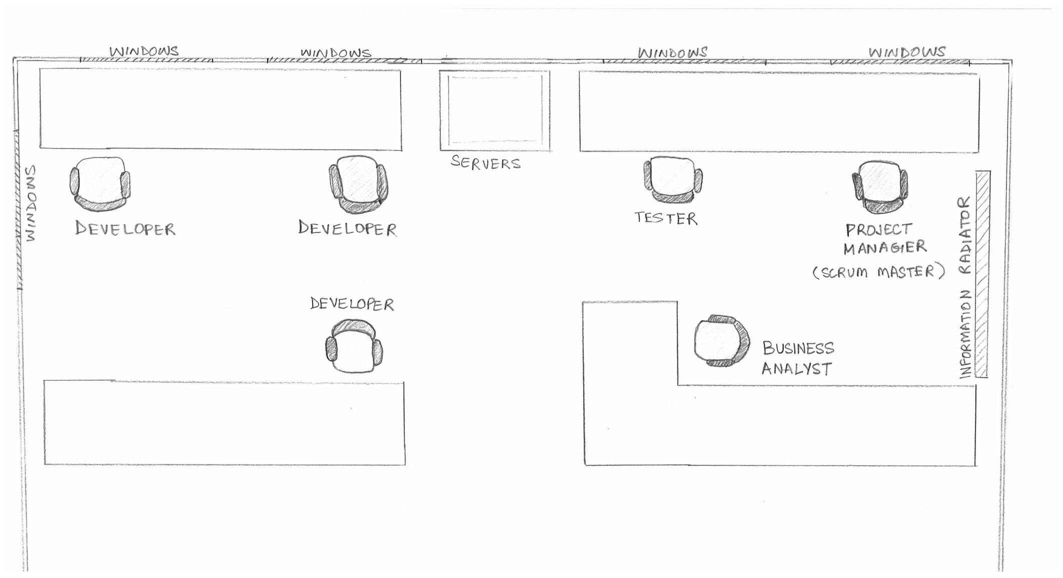


Figure 3.2: Physical setup of an open-plan workspace

Observation of an Open-Plan Workspace, New Zealand

"The office was an open planned one...The project team was located at one end of the floor and the area was occupied by five employees. These were: P6, the scrum master, one BA, three developers (from XYZ company) and a tester. The tester was on leave that day on account of an injured wrist. Her absence had started to show effects on the burndown chart already! P6 introduced me to the team members and I took the opportunity to request interviews with a couple of the experienced ones...then had a look at the white board—the information radiator. It contained story cards with point estimations. The team had a fun way of displaying ownership of tasks through cartoon characters. Each member had printed out a small-sized cartoon character, which was stuck onto a magnet and moved around with their respective

> *tasks. I thought this was a fun way to not only show ownership but also personalize the task. There were burndown charts on the white board and electronic copies of most of this information was available as shared excel files which were accessible by the whole organization. I asked P6 whether the Product Owners checked out their white boards and charts etc and it seemed like most of them were not as involved as P6 would have liked. She did mention one Product Owner flying down to discuss the charts/tasks and was fascinated by the concepts.*"

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 semi-structured 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 hand written notes were taken. Although Glaser advises against it, voice recording the interviews helped avoid losing information, and enabled better concentration on the conversations. The interview transcripts served as a good starting point for analysis. 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.

3.4.4 Data Analysis

Data analysis—called *coding* in GT—can begin as soon as some data has 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*" [63]. 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*" [63]. 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.

Open Coding

Open coding is the first step of data analysis. Open coding was used to analyze the collected data in detail [58, 60]. 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 [78].

Open coding begins by collating key points from each interview transcript. Then a code—a phrase that summaries the key point in 2 or 3 words—is assigned to each key point [57]:

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 [7]. The use of key points made it easy to focus while coding [7].

Constant Comparison Method

The codes arising out of each interview were constantly compared against the codes from the same interview, and those from other interviews and observations. This is GT's *constant comparison method* [59, 65]. 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

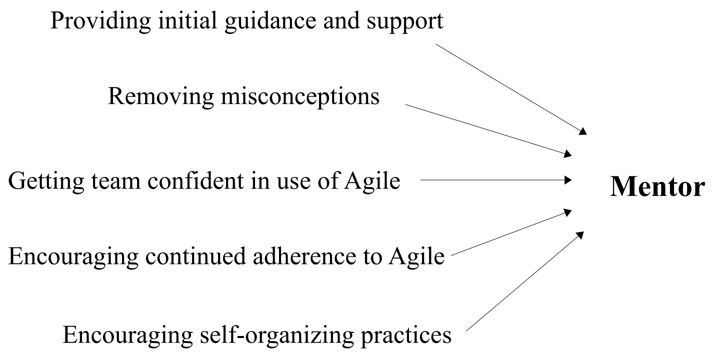

Providing initial guidance and support
Removing misconceptions
Getting team confident in use of Agile
Encouraging continued adherence to Agile
Encouraging self-organizing practices
Mentor

Figure 3.3: Example of emergence of a category from underlying concepts

Other concepts that emerged include removing misconceptions, encouraging self-organizing practices, getting the team confident in the use of Agile methods, and encouraging continued adherence to Agile. 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 that assumes the responsibility of providing guidance on the chosen Agile method. Detailed description of the *Mentor* role and other roles is presented in chapter 4. Figure 3.3 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 derived from [7, 57].

Figure 3.4 depicts the levels of data abstraction in GT [81]. Other codes, concepts, and categories emerged in a similar manner. Emergence of the different categories is presented in similar diagrams throughout this thesis.

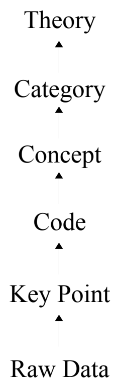


Figure 3.4: Levels of Data Abstraction in Grounded Theory [81]

The rigour 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 existing category or new properties of an existing category is discovered leading the researcher to revisiting 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.

The challenge for a software engineering (SE) researcher in applying open coding is that deriving codes, concepts, and categories, can be difficult especially early in the project. This problem was overcome by thinking of the constant comparison method as a model for data abstraction and normalization. Once the constant comparison method was understood as analogous to software engineering's method of abstraction, it became easier to apply. One of the advantages of an SE researcher using GT is that they are well-trained in analytical thinking and abstraction. The ability to raise concepts to higher levels of abstraction is something SE researchers are familiar with. This ability was extremely relevant when employing GT's constant comparative method.

An SE researcher can also become overwhelmed as raw data gets converted to another set of data (codes). The growing number of interviews means increasing amounts of codes which can be further confusing. The strategy found useful when conducting open coding was asking some questions: [60]: "*what is this data a study of?*", "*what category does this incident indicate?*", "*what is actually happening in the data?*", "*what is the main concern being faced by the participants?*" and "*what accounts for the continual resolving of this concern?*" Answering these questions allowed coding to continue effectively without feeling overwhelmed by the data.

Some GT researchers use software research tools such as NVivo [121] to conduct their analysis [126]. The use of NVivo was attempted, but its structural framework was found to limit the way data could be organized. The process of coding with pen along paper margins was found most effective. The codes, concepts, and categories were then stored into electronic spreadsheets, along with a list of the interviews or observation they were derived from. The use of spreadsheets provided greater freedom in organizing the data. As more data was collected, previous data were revisited and compared to the new ones, in-keeping with the constant comparison method. This resulted in several passes of coding and constant comparison over the entire data set.

Core Category

The end of open coding is marked by the emergence of a *core category* [59]. The core category "*accounts for a large portion of the variation in a pattern of behaviour*" and is considered the "*main concern or problem*" for the participants [58].

There are several criteria for choosing the core category: it must be central; it must be related to several other categories and their properties; it must re-occur frequently in the data; it must relate meaningfully and easily with other categories; and it must account for most variations in data [58]. The category that passed all the criteria for core was *self-organizing Agile* *teams.*

The core category captures the main concern of the participants, which becomes the research problem. A challenge for the researcher, however, is that discovering a core category can be time consuming and tedious. In absence of a core category, the researcher can easily feel confused and lost. Trusting a core category to emerge is perhaps the most demanding part of the whole GT process. The solution is to continue patiently and rigorously with constant comparisons and writing of theoretical memos (explained in section 3.4.5) and as Glaser reassures enumerable times, "*it just has to emerge*" [58]. The "eureka moment" experienced when discovering the core is truly worth the patience and toil.

Another challenge is the difficulty in discerning the core from near-core categories. For about half way through the research, the category *lack of customer involvement* was one of the most common concerns of the participants and looked promising to be the core. The solution to expose red-herrings (a near-core category appearing to be the core category) is to return to the list of criteria governing the core category. In checking the category *lack of customer involvement* against the core criteria list, it did not meet all the criteria, in particular it didn't account for most variations in data. It became apparent that *lack of customer involvement* was not the core category, rather one of the challenges faced by Agile teams in resolving their main concern, the core category: *self-organizing Agile teams.*

Selective Coding

Once the core category is established, 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*" [58, 62]. The core category guides further data collection, analysis, and theoretical sampling [58].

Selective coding was much easier compared to open coding for three reasons: (a) by the time the selective coding stage was reached, the constant comparison method had been familiarized (b) confidence in the application of GT in general was better compared to the start of the research (c) it was much easier to code selectively for only those categories that related to the core rather than continue coding for all categories.

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* [59]. 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 a clear indication of theoretical saturation.

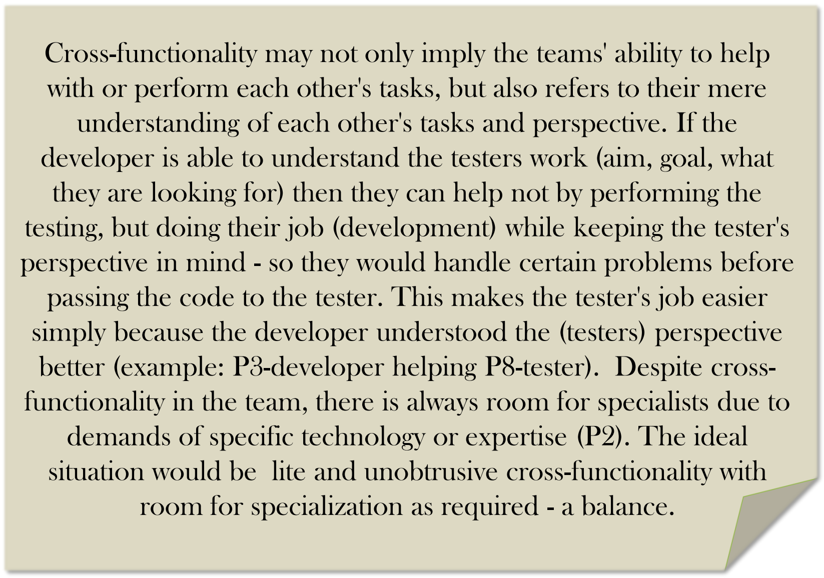
3.4.5 Memoing

Memoing is the ongoing process of writing theoretical memos throughout the GT process. A theoretical memos is a "*theoretical note about the data and the conceptual connections between categories written down as they strike the researcher*" [58]. Memoing is considered "*the bedrock*" of theory generation [58].

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. As recommended by Glaser, coding and other activities were often interrupted to capture ideas into a memo. Figure 3.5 shows an example memo on "*cross-functionality*".

Memoing is a 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.

The challenge for SE researchers in this procedure of GT is that they may not be able to express their ideas well enough in writing. A natural inclination



Cross-functionality may not only imply the teams' ability to help with or perform each other's tasks, but also refers to their mere understanding of each other's tasks and perspective. If the developer is able to understand the testers work (aim, goal, what they are looking for) then they can help not by performing the testing, but doing their job (development) while keeping the tester's perspective in mind - so they would handle certain problems before passing the code to the tester. This makes the tester's job easier simply because the developer understood the (testers) perspective better (example: P3-developer helping P8-tester). Despite cross-functionality in the team, there is always room for specialists due to demands of specific technology or expertise (P2). The ideal situation would be lite and unobtrusive cross-functionality with room for specialization as required - a balance.

Figure 3.5: Memo on *Cross-functionality*

towards literature was an advantage because I was used to writing articles, poems, and stories, which are all forms of articulating ideas into words. A SE researcher with little knowledge or inclination towards writing, on the other hand, could think of memoing as ‘thinking aloud’. Format, structure, spelling, or style etc are not to be bothered about, instead memoing should focus on getting ideas down. For example, note the spelling of ‘lite’ towards the end of the memo on cross-functionality in Figure 3.5.

Another related challenge is that memoing can easily become a trivial exercise in tracing where the codes originated [7]. A way to overcome this problem is by avoiding writing about the participants, and instead focusing on the codes and concepts. For example, the memo in Figure 3.5, does refer to some participant identifiers only as a reminder of their context. The main focus of this memo is the concept cross-functionality.

It is useful to record memos electronically on the computer so they can be stored, searched, retrieved, and edited with greater ease than using pen and paper. Separate files for memos on different topics were created and saved using the topic name for easy recall. This also supported *sorting*.

3.4.6 Sorting

Once data collection is nearly finished and coding is almost saturated, the researcher can begin sorting the theoretical memos. Sorting the memos forms a theoretical outline. Sorting is an "*essential step*" that "*can't be missed*" [58]. The advantage of sorting is that it "*puts the fractured data back together*" [58]. 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.

Printouts of all the memos were taken. They were sorted by their topics so that related topics were ordered one after the other. An outline of the theory was generated, using these topic names in the same order. This outline later formed the outline of this thesis.

The challenge involved in sorting the memos is that while it is easy to group together related memos, the ordering of the memos may not be immediately obvious. It takes some shuffling around of memos and thinking out the relationships between the different memo topics, to find an order that makes most sense. Modeling relationships between the different categories with pen on paper was found to be useful. Once the relationships were established in a diagram (using lines to connect categories), it was easier to spot how the memos (covering different categories and concepts) were related.

3.4.7 Major Literature Review

Once the findings seemed sufficiently grounded and developed, the literature on self-organizing Agile teams was reviewed. The purpose of major literature review *after* analysis is to (a) protect the findings from preconceived notions and (b) to relate the research findings to the literature through integration of ideas [58].

The advantage of literature review in later stages of GT is that it allows the researcher to spot literature that is related to the already developed concepts and categories of the emerging theory. Personal experience suggests another advantage of avoiding extensive literature review up front, namely, participants often feel more comfortable in expressing their honest opinions and sharing their real experience when informing a novice, rather than when being interrogated by an expert.

This thesis provides a literature review in chapter 2 for the benefit of the reader, however, most of the extensive reviews were conducted towards the end of the research, tieing the results into existing literature. In keeping with the order of the major literature review, the research results are followed by a discussion of existing literature. For example, the result chapters–4, 5, and 6—first present the research findings and then discuss them in relation to existing literature in a discussion section.

3.4.8 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.*" [59]. Theoretical coding involves conceptualizing how the categories (and their properties) relate to each other, and how they can be integrated into a theory [58].

Glaser lists several common structures of theories known as theoretical coding families [59, 63]. Some of these include: The Six C's (causes, contexts, contingencies, consequences, covariances, and conditions); Process (stages, phases, passages etc); Degree family (limit, range, intensity, etc); Dimension family (dimensions, elements, divisions, etc); Type family (type, form, kids, styles, classes, genre) and many more. Although theoretical codes are not strictly necessary, but "*a GT is best when they are used.*" [63].

Following Glaser's recommendation, theoretical coding was employed at the later stages of analysis, rather than being enforced as a coding paradigm from the beginning [59, 63]. The theoretical coding family found best fit to describe our findings on self-organizing Agile Teams was the Models family [58]. The Models family allows a GT researcher to model their theory diagrammatically. A figure modeling the theory is captured in figure 4.2.

3.4.9 Write-up

Following the GT method led to the generation of a substantive grounded theory of self-organizing Agile teams. The final step in GT is writing up the theory, which follows the theoretical outline generated as a result of sorting and theoretical coding. We present our write up in chapters 4, 5, and 6.

3.4.10 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 [59].

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*" [60]. In other words, an emerging theory is said to 'fit' if it explains and fits the experiences of participants as well as different practitioners who were not involved in theory generation [117].

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*".

Relevance is achieved when the criteria of fit and work are met. Relevance evokes instant "*grab*" [60].

Modifiability is a "*quality of the theory to be ready for changes to include variations in emergent properties and categories caused by new data.* " [60].

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

These criteria are revisited at the end of this thesis in section 7.3 to demonstrate how well our theory evaluates against them.

3.5 Discussion

This section captures some reflections on the application of GT to study software engineering, and in particular, Agile software development teams.

Through the course of our research, a strong synergy between the research area (Agile software development) and the research method (Grounded Theory), were discovered [81]. There are several commonalities between the two: both advocate minimum initial planning—Agile methods advocate minimum design and planning up-front while Grounded Theory recommends minimum initial literature review; both are iterative and incremental in nature—Agile methods have set iterations in which the teams develop small chunks of working functionality towards the final product while the Grounded Theory method involves iterative rounds of data collection and analysis (albeit of flexible lengths) such that each iteration brings the researcher a step closer to the main concern of the study; both focus on the human and social aspects—Agile methods value "*people and interactions over processes and tools*" [72] while GT focuses on studying the human experience and social interactions in a given substantive area. Applying GT requires the ability to embrace uncertainty, as the research focus slowly emerges through iterative rounds of data collection, analysis, and memoing. This is similar to Agile software development's dictum of "*embrace change*" and "*responding to change*" [19, 72]. The ability to embrace this uncertainty is somewhat dependent on the researcher's personality. Some researchers may find this extremely uncomfortable and become paralyzed, while others feel excited at the prospect of chasing and discovering the hidden or the unknown. The key, as Glaser relentlessly repeats, is to return to data and trust emergence.

Some researchers feel that it is nearly impossible to let the research question emerge in the process of conducting GT [149]. Avoiding extensive literature review up-front and trusting the emergence of core concern make such skeptics nervous. Our own experience of using GT as a research method in a SE area with no previous theoretical training in GT to begin with is a demonstration of an application of GT. Emergence can happen as long as the fundamental tenets of the methods are adhered to and the researcher is able to use theoretical sampling effectively to continuously narrow the focus of the study to a single most relevant topic or concern. Our application of Grounded Theory to SE research was not smooth-sailing, as is evident from the various challenges faced (and described) in each of the GT procedures. The strategies found useful in overcoming these challenges, however, infuse confidence in employing GT again for similar studies in the future. The description of the challenges faced and the strategies found useful in applying GT should help other SE researchers attempting to use GT.

Chapter 4

Self-Organizing Agile Team Roles

This chapter presents the core of our grounded theory of self-organizing Agile teams. The theory explains how software development teams take on informal, implicit, transient, and spontaneous *roles*; perform balancing acts on a set of integrated *practices*; while facing critical environmental *factors*, in order to become a self-organizing Agile team. The roles are: Mentor, Co-ordinator, Translator, Champion, Promoter, and Terminator. The practices involve balancing between freedom and responsibility, cross-functionality and specialization, and continuous learning and iteration pressure. The factors are senior management support and level of customer involvement. Each of these aspects of a self-organizing Agile team—roles, practices, and factors—are described in this chapter and the next two chapters.

Figure 4.1 shows the emergence of a grounded theory of self-organizing Agile teams from underlying categories and concepts. Figure 4.2 depicts the theory of self-organizing Agile teams as a model representing the roles, practices, and factors.

This chapter describes the informal roles that exist on self-organizing 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 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*, *Co-ordinator*, *Translator*, *Champion*, *Promoter*, and *Terminator*—are focused specifically towards self-organization. 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 4.1 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. The quotations are presented verbatim from the interview transcripts with square brackets used to insert missing words to fix grammar or to anonymize participant details (such as names of individuals or companies). Three full stops (...) indicate a pause, while three full stops preceded and followed by spaces (...) indicate combining two sentences referring to the same context but derived from different parts of the same interview.

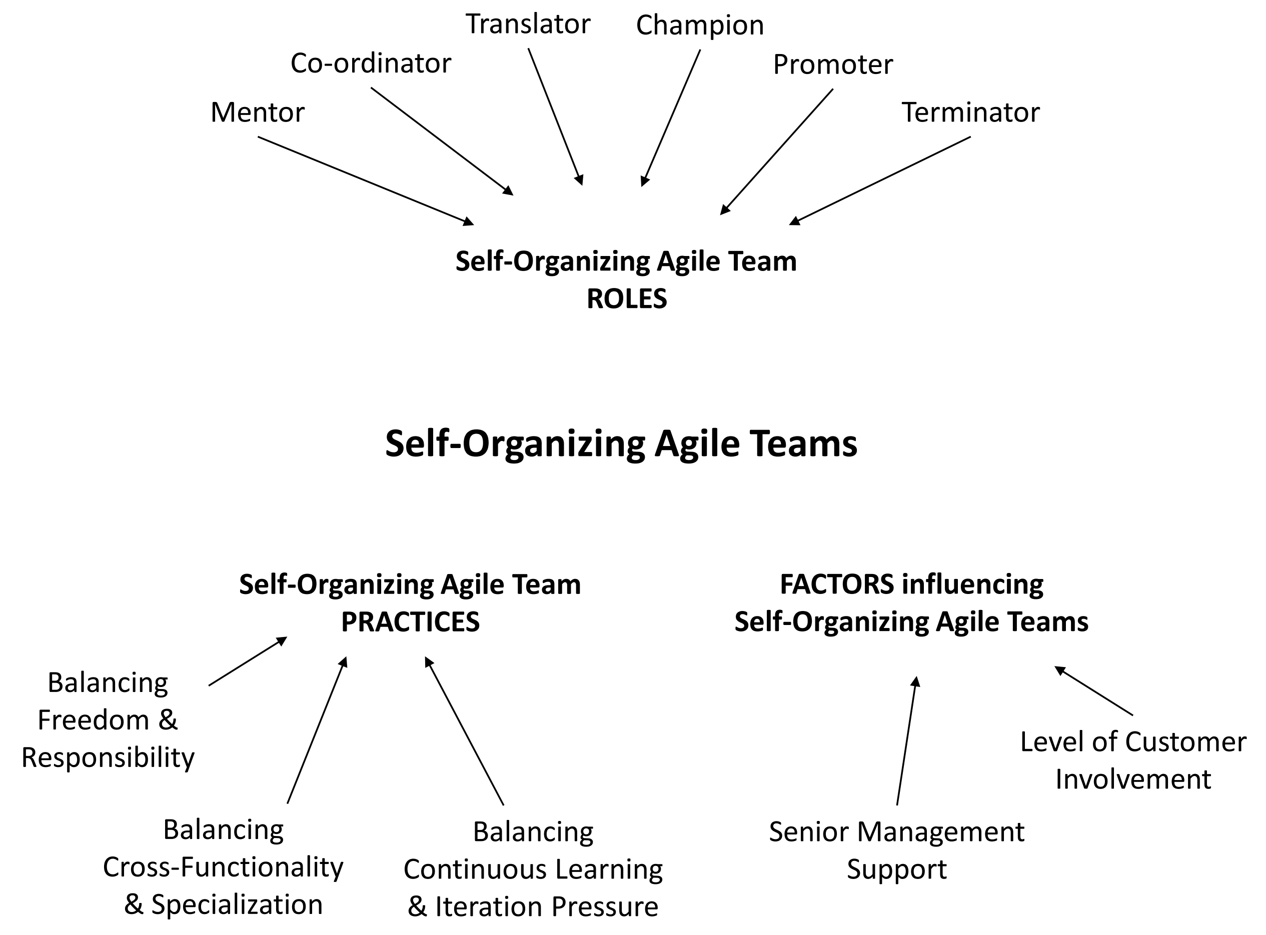

Mentor
Co-ordinator
Translator
Champion
Promoter
Terminator
Self-Organizing Agile Team ROLES
Self-Organizing Agile Teams
Self-Organizing Agile Team PRACTICES
Balancing Freedom & Responsibility
Balancing Cross-Functionality & Specialization
Balancing Continuous Learning & Iteration Pressure
FACTORS influencing Self-Organizing Agile Teams
Senior Management Support
Level of Customer Involvement

Figure 4.1: Emergence of the Self-organizing Agile Team Roles, Practices, and Factors from Underlying Categories.

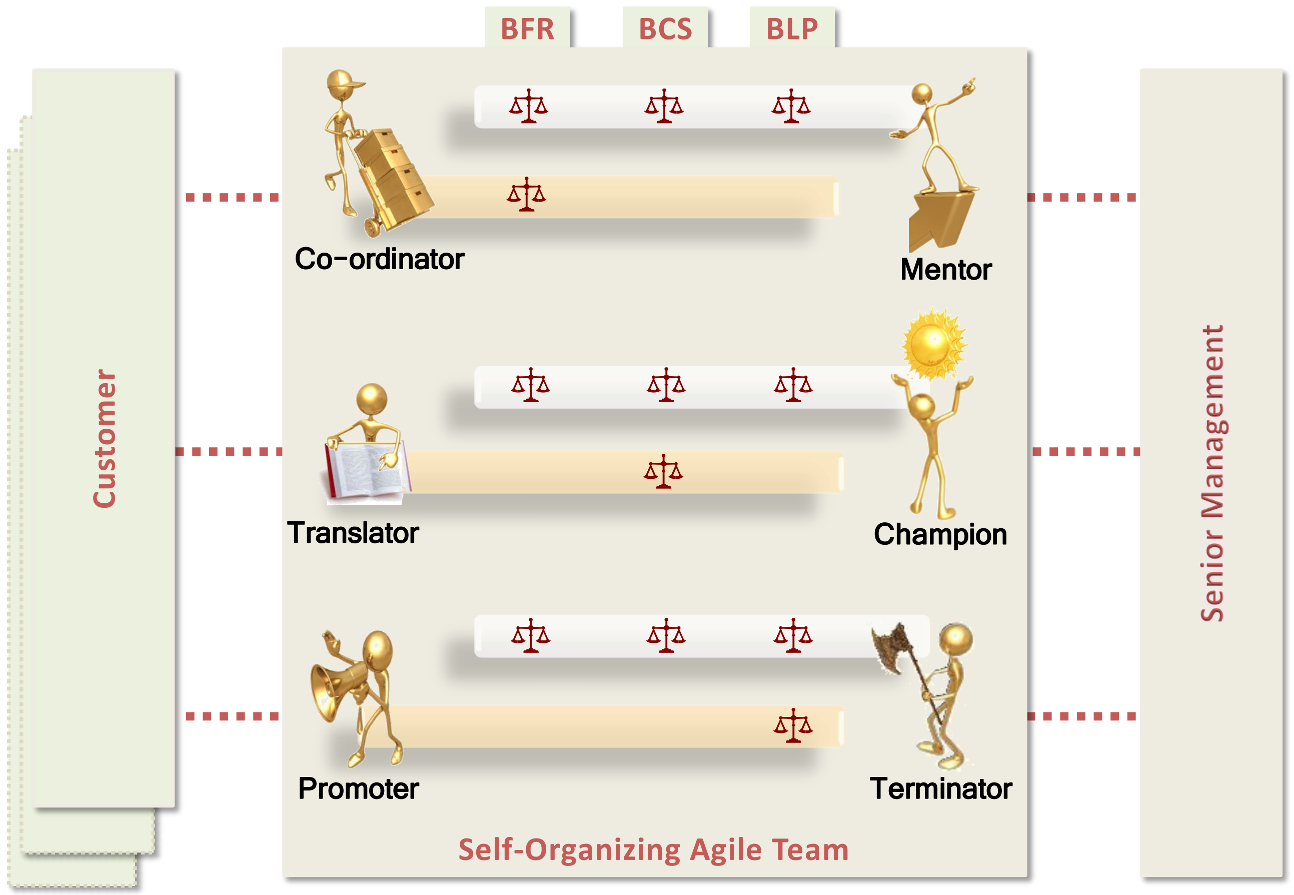


Figure 4.2: Theory of Self-Organizing Agile Teams. (Roles: Mentor, Co-ordinator, Translator, Champion, Promoter, and Terminator. Practices: Balancing Freedom and Responsibility (BFR); Balancing Cross-Functionality and Specialization (BCS); Balancing Learning and Iteration Pressure(BLP). Factors: Senior Management Support and Level of Customer Involvement.)

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 [78]. A description of how this category emerged from data analysis has been provided in section 3.4.4. Figure 4.3 illustrates the emergence of the category *Mentor* from the underlying concepts.

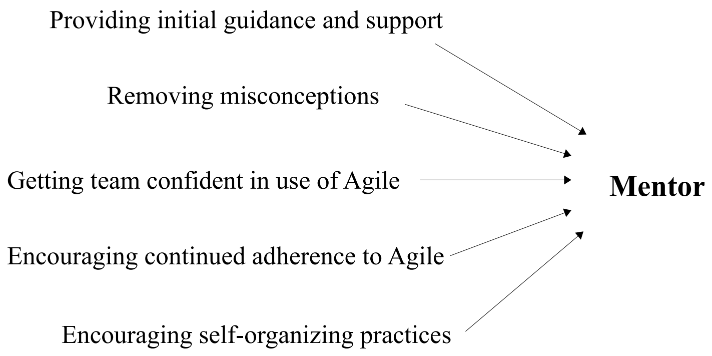

Providing initial guidance and support
Removing misconceptions
Getting team confident in use of Agile
Encouraging continued adherence to Agile
Encouraging self-organizing practices
Mentor

Figure 4.3: Emergence of the category *Mentor* from underlying concepts

Table 4.1: Self-Organizing Agile Team Roles. Agile Coach (AC), developers (Dev), business analyst (BA). Senior Management in brackets for Mentor and Terminator role indicates indirect involvement (as opposed to direct interaction)

Role	**Definition**	**Interacts with**	**Played by (in new teams)**	**Played by (in 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)

4.1.1 Providing Initial Guidance and Support

The *Mentor* familiarizes the team with the Agile Manifesto [72] 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 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 re-evaluation 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 de-*

> *fensive 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 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.

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 considerable.

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 [a] while it became natural...people were picking stuff...and that worked really well.*" — P25, Developer, New Zealand

A detailed description of the self-organizing practices is provided in chapter 5.

4.1.3 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

4.1.4 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 [a different company] 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 considerable 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 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.

4.2 Co-ordinator

Acts as a representative of the team to manage customer expectations and co-ordinate 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 [82, 74, 73]. These collaborative activities are difficult to co-ordinate with the customer for various reasons, such as physical distance between the development team and their customers, lack of time commitment on part of the customers, and ineffective customer representation [82, 74]. The *Co-ordinator* role emerged on Agile teams to overcome these challenges and facilitate collaboration with customers [78]. Figure 4.4 illustrates the emergence of the category *Co-ordinator* from the underlying concepts.

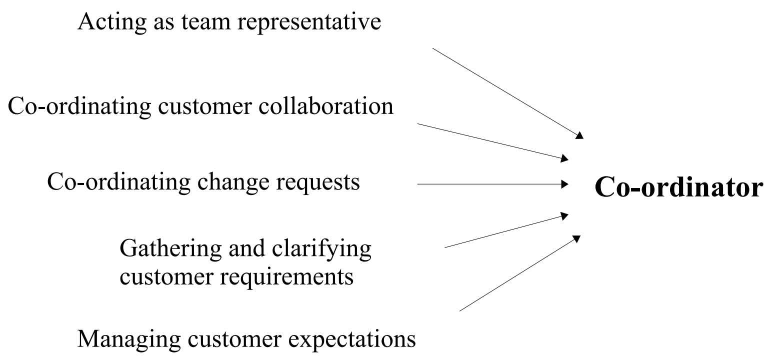

Acting as team representative
Co-ordinating customer collaboration
Co-ordinating change requests
Gathering and clarifying customer requirements
Managing customer expectations
Co-ordinator

Figure 4.4: Emergence of the category *Co-ordinator* from underlying concepts

4.2.1 Co-ordinating Customer Collaboration

In the context of the Indian software industry, Agile teams often face off-shored customers. Co-ordinating 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** co-ordinating between the team and their distant customers representatives. In one of the Indian projects, the *Co-ordinator* role was played by a developer who helped co-ordinate 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 *Co-ordinator* interacts with the team on a regular and intimate level and co-ordinates 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 *Co-ordinator* to facilitate customer collaboration. In case of a New Zealand team, a business analyst on the team acted as the *Co-ordinator*, representing the team to their customers and co-ordinating 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

A *Co-ordinator* is useful in situations where the customer representative is unable or unwilling to devote the amount of time that the teams require to collaborate [74, 82]. Similarly, the *Co-ordinator* role helped facilitate collaboration with customer representatives that the teams perceived to be largely ineffective.

> "*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). 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.

4.2.2 Co-ordinating Change Requests

The *Co-ordinator* also helps co-ordinate change requests made by the customers. Responding to change [72, 100] is an integral part of Agile methods and a *Co-ordinator* 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 *Co-ordinator* 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 *Co-ordinators*, co-ordinating 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.*"

Observing a *Co-ordinator* in action supplemented the data derived from interviews and strengthened the understanding of the role. When asked about these *Co-ordinators*, 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 co-ordinating across distances. These two developers were better communicators compared to the rest of the team and had spontaneously taken on the *Co-ordinator* 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 *Co-ordinator* 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 *Co-ordinator* 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 *Co-ordinator* role was played by a developer that interfaced with off-shored customers on behalf of the team. On a relatively new New Zealand team, the *Co-ordinator* 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 *Co-ordinator* 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 playing the *Co-ordinator* role and co-ordinate 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 *Co-ordinator* role exists despite the presence of the *Mentor*.

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 [74, 80, 82]. 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 on self-organizing Agile teams to overcome the language barrier [74, 80, 82]. Figure 4.5 illustrates the emergence of the category *Translator* from the underlying concepts.

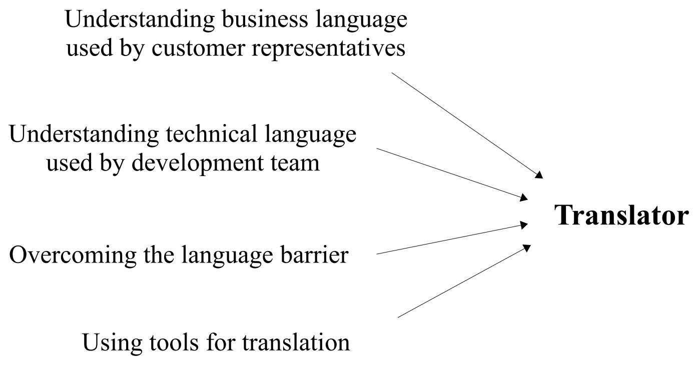

Understanding business language used by customer representatives
Understanding technical language used by development team
Overcoming the language barrier
Using tools for translation
Translator

Figure 4.5: Emergence of the category *Translator* from 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*' language with domain specific requirements. The development team need 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. On other more mature teams, 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

Some participants ensured that they were "*hiring smart, pragmatic communicators*" (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* role [80]. These include: a project dictionary, using iterative reasoning, and encouraging cross-functionality in the team.

One of the Indian teams use 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 [154]. 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 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 *Co-ordinator* 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 *Co-ordinator* 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 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 [78].

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 [83]. The *Champion* role emerged on Agile teams to secure senior management support [78]. Figure 4.6 illustrates the emergence of the category *Champion* from the underlying concepts.

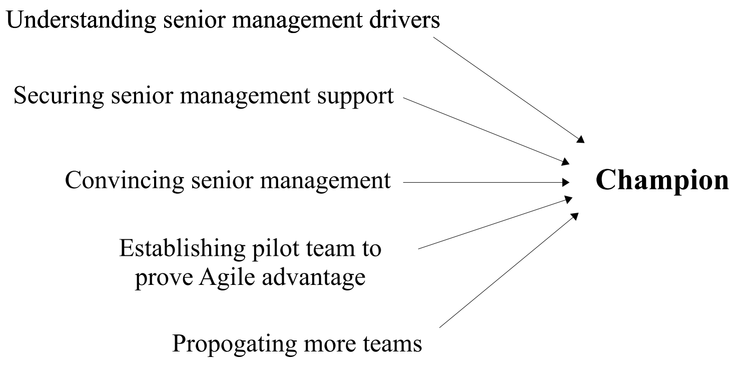

Understanding senior management drivers
Securing senior management support
Convincing senior management
Establishing pilot team to prove Agile advantage
Propogating more teams
Champion

Figure 4.6: Emergence of the category *Champion* from 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 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

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

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:

> "*The [Champion] was pretty much championing the whole Agile idea. They were thinking of using [the Champion] to expand Agile through all of [organization], so every single project they were looking at trying to put an Agile aspect to it and [the Champion] was doing all the ideas, all the objective identification, everything*" — P4, Business Analyst, 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 champions 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 favour the proper functioning of self-organizing Agile teams. The impact of senior management on self-organizing Agile teams is discussed further in chapter 6.

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 influence 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 part of the customers, etc. The *Promoter* role emerged to overcome the lack of customer involvement [74, 82]. Figure 4.7 illustrates the emergence of the category *Promoter* from the underlying concepts.

4.5.1 Understanding Customer Concerns

Customers can harbour 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*

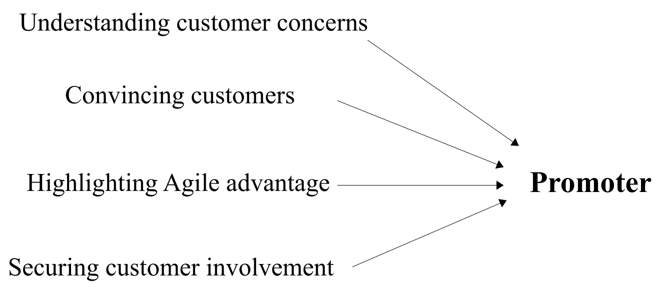

Understanding customer concerns
Convincing customers
Highlighting Agile advantage
Securing customer involvement
Promoter

Figure 4.7: Emergence of the category *Promoter* from underlying concepts

> *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 hallabalu about Agile this, Agile that! we showcase our [unique] offering, we showcase case studies and also give them a sense of—not lolling 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 [82]. 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 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 *Promoter* and *Co-ordinator* 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 *Co-ordinator* 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. Figure 4.8 illustrates the emergence of the category *Terminator* from the underlying concepts.

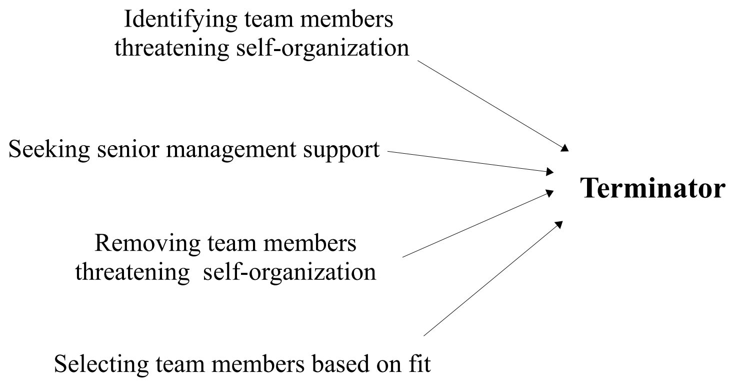

Identifying team members threatening self-organization
Seeking senior management support
Removing team members threatening self-organization
Selecting team members based on fit
Terminator

Figure 4.8: Emergence of the category *Terminator* from underlying concepts

The role of the *Terminator* is certainly not an easy one, and perhaps the most controversial.

4.6.1 Identifying Threatening Team Members

The *Terminator* identifies individuals in the team that may be hampering team productivity because of their personal characteristics and practices. Individual personality of team members can be considered more important than skill set when selecting an Agile team. As one of the *Terminators* acknowledges below, the individuals themselves are not "*bad*", but that their personality is 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 towards 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, ability to communicate, 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 don't think that's 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 fallback 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 up front.

> "*[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. In mature teams, the *Terminator* role was played by an Agile coach supported by the rest of the team. 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 Resource—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

4.7 Role of the Agile Coach

The self-organizing Agile team roles identified in this research make an Agile team self-organizing. This leaves a critical question unanswered: what is the role of the Agile coach on a self-organizing Agile team? As one of the participants noted:

> "*Actually if you talk to some people who are new in the Scrum Master role, they ask: 'what is our job?' If you tell them you resolve impediments, they understand it but how do you apply it to reality?*" — P31, Agile Coach, India

An important contribution of this thesis is to define the role of an Agile coach, in terms of the self-organizing Agile team roles they are likely to play at different stages of the teams' maturation. The Agile coach is either played by contracting consultants or by an existing project manager within the organizations. In the latter case, the person playing the Agile coach may still keep their formal organizational title of manager, or project manager.

In relatively new Agile teams, the role of the Agile coach is extremely important. Initially, an Agile coach takes on most of the self-organizational roles discovered in our research in an effort to facilitate self-organization in the team in the early stages. On a new team, an Agile coach is likely to play any or all of these roles: a *Mentor* to train the new team on Agile principles, values, and practices; a *Co-ordinator* to co-ordinate customer collaboration; a *Translator* to help translate business specifications into technical requirements for the new team; a *Champion* to gain senior management support for the team; a *Promoter* to secure customer involvement for the proper functioning of the team; and a *Terminator* to remove cultural misfits from the team.

> "*If you put the project manager in that role you'll find that the team would grow into a self-organizing team. That's where the real power comes into the picture. He should not be interfering into the day to day activities of the team: [what] is to be done, [what] is the priority of changing things...generally I see that a typical project manager become a team coach for an Agile team—because [for] new teams if you chose someone from inside the team he doesn't have that kind of mindset to act as a real Scrum Master. So I see that a Project Manager should get transformed into that role because he's sort of suited for it; that's his job.* ***If you can separate out the micro-management, then [the PM is] the ideal Scrum Master for an Agile project.***"
> — P31, Agile Coach, India

Over time, these self-organizing team roles are taken up by the team members. And so, in more mature Agile teams, most members of the team have the caliber and experience to play any of the roles. For example, in mature teams, the *Mentor* role is often played by experienced team members that help mentor newcomers on the team; the *Co-ordinator* and *Translator* roles are played by most members of the team as they gain experience in collaborating directly and frequently with their customers; the *Champion* and *Promoter* roles are played, as required, by more experienced members of the team. The *Terminator* role is played by the Agile Coach with support from the team as they provide their input into the suitability of an individual to join or remain in an Agile team (section 4.6).

This suggests that the role of an Agile coach is to play most of the self-organizing Agile team roles initially and gradually pass them on to the team members. In other words, as a couple of the participants noted:

"A PM's [Project Manager's] job is to make himself or herself redundant. So then the team is self-organized, everybody is accountable... PM doesn't have to do much, everything is in place and now I can go and do something else...I want to do some enabling, some team building...making sure all the processes are in place. " — P47, Business Analyst, India

"...project managers...are there with the specific purpose to serve and protect the teams and to ensure the project is in good health ... Analogy is you have patient on the bed, there are all these things connected to that person; the doctors don't check each and everything piece of equipment...[they] look at the status, graph looks good, good system. Now that is all a manager is doing, somebody who's there when things really break down, when there is a better equipment out there and a better means of ensuring the systems are functioning, that is what the manager should be doing...[an] advantage of Agile is that it takes away all [this micromanagement], brings in all the self-monitoring, self management, this higher levels of commitments and responsibility. Instead of concentrating power in one resource...you are just distributing the load onto the relevant forces and you're just focusing what a management should be—core issues, what is the strategic partnership, decisions being made." — P36, Agile Coach, India

4.8 Discussion

Following Grounded Theory, the data was 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 to (a) protect the findings from preconceived notions and (b) to relate the research findings to the literature through integration of ideas

[58]. This section discusses our results in the light of related literature [1].

4.8.1 Team Roles

A wide number of researchers have explored Team Roles and Dynamics [9, 22, 40, 47, 113, 134].

Belbin suggests nine team roles based on behaviour: plant, resource investigator, co-ordinator, shaper, monitor evaluator, teamworker, implementer, completer finisher, and specialist [22]. A co-ordinator in Belbin's team roles theory focuses on team's objectives and delegates work. The *Co-ordinator* role identified in our research, on the other hand, helps co-ordinate 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. This practice of a self-organizing team is discussed in detailed in the next chapter.

Five boundary-spanning roles have been identified as means to encourage communication across boundaries: ambassador, scout, guard, sentry, and co-ordinator [9, 134]. 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

[1]In this section, the term "our" is used to refer to this thesis, to differentiate this research from the related literature being discussed.

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 pro-actively seek the support of their environmental factors through *Champion* and *Promoter* roles.

Anderson et al. [10] define self-organizing teams as teams that are (a) informal and temporary, (b) formed spontaneously around issues (c) are not a part of a formal organization structure, (d) possessing a strong sense of shared purpose, (e) where team members decide their own affairs, and (f) where all members' primary roles relates to the task. The roles identified in this research (*Mentor*, *Co-ordinator*, 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 [10]. In other words, these informal, implicit, transient, and spontaneous roles make Agile teams self-organizing [78].

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. Details of practices that enable transparency in Agile teams are presented in chapter 5. The last of the five boundary-spanning roles is the co-ordinator. Much like the *Co-ordinator* role identified in our research, this co-ordinator role focuses on communication with external groups while keeping them informed of the team's progress.

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 [47]. Our *Mentor* role is the closest to the classic Agile coach described in the Agile literature [16, 111, 128, 138].

Some studies have described individuals supporting customers by translating *technical* language to *business* language [104, 107]. 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 [40] recommend placing "*more emphasis on people factors in the project: amicability, talent, skill, and communication.*" In our research, practitioners used their own set of criterion 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 did not fit in with the rest of the team, and hampered their productivity due to lack of openness and willingness to change.

4.8.2 Role of the Agile Coach

Self-organizing teams are not meant to be leaderless and uncontrolled [40, 154]. Leadership in self-organizing teams is meant to be *light-touch* and *adaptive* [16], providing feedback and subtle direction [11, 34, 154]. This is in contrast to centralized management in traditional teams [30, 3, 2]. Leaders of Agile teams are often compared to coaches of sports teams—responsible for setting direction, aligning people, obtaining resources, and motivating the teams [11]. Agile methods, such as Scrum and XP, define the Scrum Master or XP coach (referred to as the Agile coach in this thesis) as a facilitator of the self-organizing Agile team [19, 138]. According to the Scrum and XP guidelines, a Scrum Master is responsible for protecting the team from any disruptions to their tasks that may be caused by outside sources [113, 128, 138], such as unrealistic demands from the customers. They ensure that the team is fully functional and productive and that all Scrum processes are being followed. A Scrum Master is seen as a facilitator and does not organize or manage the team [138]. Similarly, an XP coach is meant to lead the team towards self-organization by leaving the team alone as early as possible [56]

Despite the guidelines laid down by Agile Methods [19, 138], the role of an Agile coach is one of the most popular topics of debate among industry practitioners. Inexperienced Agile coaches, as well as experienced project managers used to a traditional development environment, find themselves confused when they start practicing Agile methods.

Books have been written by experienced practitioners that acknowledge the predicament faced by new Agile Coaches in understanding their role and offer advice from practical experiences [16, 93, 144]. Sanjiv Augustine and Susan Woodcock explore the role of the project manager and propose the concept of visionary leader as opposed to an uninspired taskmaster [16]. While traditional management was viewed as governing and commanding, experienced Agile project managers are meant to display 'light touch' leadership [16]. Similar sentiments are resonated by Mary Poppendieck in a panel discussion titled *Agile Management An Oxymoron?* notes "*I distinguish management tasks getting the maximum value from the dollar—from leadership tasks—helping people to excel. Leaders are required. Managers are optional*" [11].

Research on the role of an Agile coach is extremely limited. Coram and Bohner have studied the impact of Agile methods on software project management and touched briefly on the project manager role in Agile. They noted that the project manager is a much more "*involved role*" and that "*project managers in agile processes are responsible for tracking progress and making business decisions*" [44]. A change in the role of the traditional manager has been predicted [119]. Our research helps define the role of an Agile coach on self-organizing Agile teams (section 4.7).

This chapter has described (a) these are informal, implicit, transient, and spontaneous self-organizational roles on Agile teams, discovered through this research: *Mentor*, *Co-ordinator*, *Translator*, *Champion*, *Promoter*, and *Terminator*; (b) a mapping between the self-organizational roles and their organizational roles associated with the individuals who played them; (c) a description of the role of an Agile coach in terms of the self-organizational roles they play on Agile teams; and (d) a discussion of the roles in light of existing literature.

Chapter 5

Self-Organizing Agile Team Practices

Chapter 4 described the informal **roles** that facilitate self-organization in Agile teams. This chapter presents the **practices** that enable self-organization in Agile teams—"*the balancing acts*" [76]. The term "*balancing acts*" emerged from the data analysis, as shown in Figure 5.1, to describe the practices of self-organizing teams that balance between different (and often contrasting) concepts.

The balancing acts include several low-level practices that enable self-organization on an every day basis. Balancing freedom and responsibility involves practices such as collective decision making through collective estimation and planning, collectively deciding teams and principles, and self-committing to team goals; self-assignment using story boards; self-monitoring through daily standup meetings and use of information radiators. Balancing cross-functionality and specialization involves practices such as multiple perspectives, group programming, rotation. Balancing continuous learning and iteration pressure involves practices such as retrospectives, learning spike, and pair-in-need. Table 5.1 shows the Agile practices that specifically enable self-organization on Agile teams grouped under their corresponding balancing acts.

The following sections describe self-organizing Agile team practices. A discussion of how the balancing acts support and complement each other, how they relate to the general principles and specific conditions of self-organization, and how they relate to other relevant literature concludes the chapter.

Table 5.1: Self-Organizing Agile Team Practices

Balancing Freedom & Responsibility
Collective estimation and planning
Collectively deciding team norms and principles
Self-committing to team goals
Self-assignment using story boards
Self-monitoring through daily standups and information radiators
Balancing Cross-Functionality & Specialization
Multiple perspectives
Group Programming
Rotation
Balancing Continuous Learning & Iteration Pressure
Self-evaluation through retrospectives
Self-Improvement through learning spike and pair-in-need

5.1 Balancing Freedom and Responsibility

Team members experience more freedom as a part of a self-organizing Agile team than as a part of a traditional software development team. Managers on traditional teams are responsible for setting team goals, assigning individual tasks for the team members to achieve within set time-frames, and micromanaging the projects on a daily basis (P10, P20, P36, P56). Agile team members with previous experience of working in traditional software

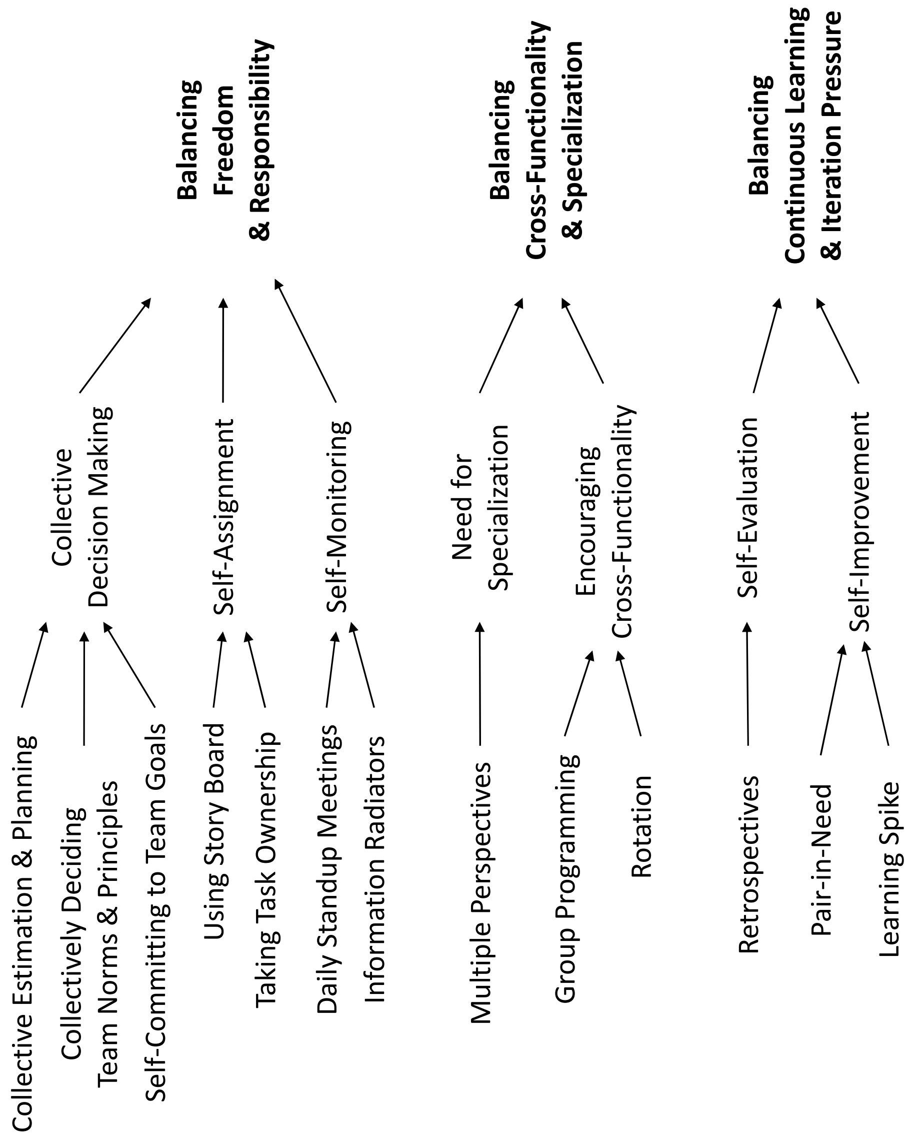

Collective Estimation & Planning
Collectively Deciding Team Norms & Principles
Self-Committing to Team Goals
Collective Decision Making
Using Story Board
Taking Task Ownership
Self-Assignment
Daily Standup Meetings
Information Radiators
Self-Monitoring
Balancing Freedom & Responsibility
Multiple Perspectives
Need for Specialization
Group Programming
Rotation
Encouraging Cross-Functionality
Balancing Cross-Functionality & Specialization
Retrospectives
Self-Evaluation
Pair-in-Need
Learning Spike
Self-Improvement
Balancing Continuous Learning & Iteration Pressure

Figure 5.1: Emergence of the category *Balancing Acts* from underlying concepts

development teams describe a traditional environment as frustrating and demotivating.

> "*[In traditional projects] it was more demotivating to be given ridiculous deadlines or just feel that the people [managers]...who are deciding the deadlines don't actually have any clue about the technical challenges associated with them.*" — P11, Developer, NZ

In contrast, Agile teams are not micro-managed by managers, rather they are provided freedom by their senior management to organize themselves. Self-organizing Agile teams perform practices that allow them to self-assign, self-commit, self-monitor, self-evaluate, and self-improve (P1-P4, P6-P7, P10-P16, P20-26, P27-29, P31-P32, P34-36, P39-40, P43-52, P54, P56), giving them a concrete sense of empowerment. The practices of collective decision making while committing and achieving team goals, self-assigning tasks, and displaying responsibility require the team to perform a balancing act between freedom and responsibility. These practices are described below, along with an example of the consequence of imbalance.

5.1.1 Collective Decision Making

Self-organizing Agile teams plan their iterations and commit to their own team goals as a result of the freedom provided by their senior management:

> "*We are participating in all the sprint planning activities and we have a clear say in that okay we'll be able to do this particular stuff in this particular sprint or we have some extra load on us or not.*" — P32, Tester, India

Self-organizing Agile teams perform **collective estimation and planning**. The customer representatives provide project requirements in the form of user stories [138]. These user stories are broken down into developmental level tasks by the teams during iteration planning meetings. The team collectively participates in estimating user stories and tasks and in planning their iterations.

Estimation and planning in self-organizing Agile teams involves everyone on the team. A typical estimation and planning session begins with the team considering the user stories provided by the customer for an iteration. In Scrum teams, estimation is done by playing *planning poker*, where every user story and task are assigned complexity points by the team on a numeric scale, depending on the team's perception of the collective effort involved in implementing them.

> "*Once we've got those tasks, we give them an estimate on how much time they'll take... We actually play a game where we all hold our fingers up to represent the number of hours we'll take, just to get away from that whole 'just following one person's idea' [in traditional development teams]. And that works quite well; a couple of times we've been to, say, look at why people are so far apart and talk it out some more and realize it's maybe not as small as one person thought it was but maybe not as big as the other person thought.* " — P13, Developer, New Zealand

In contrast, estimation and planning in traditional projects is typically done by managers and does not involve team members (P2, P36). As a result, team members with traditional software development backgrounds often have no experience in estimating and planning of projects.

> "*It was new and the first time I attended that meeting I was like what are these cards for...(laughs)...it was so confusing at the time...then I got used to what the cards meant and then later on when we had done a few months of the thing we could size something without the cards already and we already knew exactly what a size of a story is without even thinking about it because it was so natural because we'd gotten so used to it. That was fun!*"
> — P2, Developer, New Zealand

Mature Agile teams **collectively decide team norms and principles** that guide their practices. These principles include an informal understanding of working hours, team velocity, policy on defect tolerance etc.

> "*Even if all the team members are familiar with Agile, there is a stage that you have to go through. Like when we start a project we do a session called norming and charting session where we—everyone in [company name]—we all know about our Agile practices but when we start on a project we do a session where we agree to a certain set of principles. Because Agile as such doesn't dictate any core working habits like we say these are our core working hours we'd like to stick to that; this is our setup time; these are our coding practices that we'd like to adhere to; we won't have any technical debt.*" — P31, Agile Coach, India

While Agile methods grant customers the ability to prioritize user stories every iteration, the decision of how many complexity points will be attempted in an iteration (developmental pace or team velocity) rests with the team, based on their capacity. A self-organizing Agile team **self-commits to team goals** based on this velocity:

> "*We have stories which we estimate complexity of and we say 'well, we can fit this much complexity into next two weeks'*" — P10, Agile Coach, New Zealand

> "*Once we've got stories, we generally have a breakdown meeting at the beginning of each iteration, and we lay out the stories that we may not have completed and the next X many stories; we have a look at how long we originally estimated they would take and then we try and make a guess as to how many we'll do, and how many we'll get through, based on the length of time we've got and who's available. The calculation...[is] a real feeling thing based on the team.*" — P13, Developer, NZ

Self-organizing Agile teams enjoy the freedom to set their own team goals, and at the same time they realize their responsibilities to ensure that they achieve the iteration goal through a collaborative effort. The team has a strong sense of commitment to the team goal and they feel responsible to achieve it.

> "*The sprint is a commitment of the team, so if a story's not getting finished, that means somebody is not doing their job...it's a team effort as opposed to an individual effort.*" — P2, Developer, New Zealand

> "*We are given responsibility and we're given complete freedom....At the end of the day [management] wants the tasks to be done but [they] want that we do it our way. [They] have satisfaction that [we] did it in the best possible way... and if there's certain thing missing then we can just ask our friends and our colleagues whether they know a better way to do this...that's [how] we are self-organizing.*" — P44, Developer, India

Self-organizing Agile teams make collective decisions as "*every person is contributing to the decision-making*" (P20). Compared to traditional teams where the manager makes most decisions related to the team and their projects, self-organizing Agile teams make "*a lot more decisions*" collectively (P8).

> "*If the team is really at the peak of self-organization - the developers are also empowered, everybody is empowered - they can make decisions. If you don't have the Scrum Master - he's on vacation or something - then if that's not the case you'd expect everything to stop, right? but it doesn't stop - it goes on.*" — P31, Agile Coach, India

> "*they make decisions collaboratively. Nobody is standing up there and, and making a unitary decision. Where a decision has to be*

> *made, it might be made in a very short time, but it would be made with the interests of the team in mind, by everybody.*" — P26, Agile Coach, NZ

If Agile teams face a management that dictates terms and sets goals on behalf of the team, the team is unlikely to self-organize:

> "*[If] they are forced to commit to a goal that they didn't believe in - because of management pressure...if you don't give that freedom...if you have micromanagement, how can you expect people to be self-organizing? How can they take ownership of what they commit to?...[if] you have somebody from management who sits over it, who dictates it, that takes out the self-organizing nature.*"
> - P31, Agile Coach, India

Senior management within the organization must provide an environment in which teams can perform collective decision making through collective estimation and planning, collectively deciding on informal team norms and principles, and self-committing team goals.

5.1.2 Self-Assignment

Self-assignment, as opposed to delegation, is a distinguishing feature of self-organizing Agile teams. Members of self-organizing teams strongly value their ability to self-assign tasks and appreciate the freedom they have to be able to pull the tasks from the story wall and assign themselves to their chosen tasks (P1-P4, P6, P10-P16, P25-29, P31-32, P36, P39, P44-45, P58).

Committing to a team goal every iteration is a group decision. Self-assignment of tasks within committed user stories, on the other hand, is an individual decision.

> "*Agile teams its all about pull instead of push so...you will define tasks yourself and as soon as you are done with the current task,*

you pick up a new one. That's how it works." — P30, Developer, India

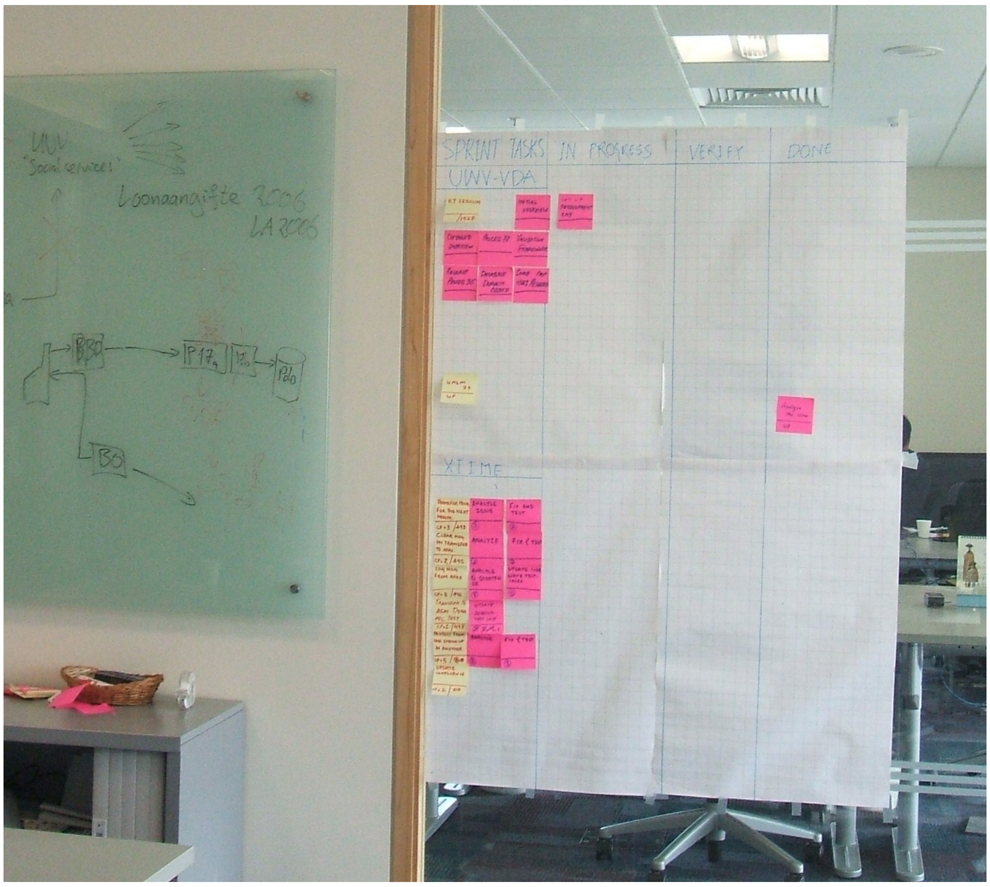


Figure 5.2: A story board/wall with user stories and tasks

A practice that enables self-assignment is the **use of story boards**. The story board (also known as a Scrum board) comprises of the user stories that the team has committed to implementing in a given iteration, along with their break-down into technical tasks, as a result of iteration planning. The user stories and tasks are written on small pieces of paper or post-it notes, and are stuck to the story board. The story board has three columns corresponding to tasks 'not started', tasks 'in progress', and tasks 'completed'. Individual teams use different terms to signify the three states. A picture of a story wall is shown in Figure 5.2, taken during observations of the workplace of an Indian team participating in the research.

The story board is placed in a visible area, such as on a wall or cabinet, for ready reference. Team members self-assign tasks by walking over to the story board and picking up a task. They physically move the task from their initial 'not started' column to the 'in-progress' column, to demonstrate that a task has been self-assigned. Self-assignment leads to **taking task ownership**. Individual teams have interesting ways of displaying task ownership. Some teams use initials of their names on the tasks, while others use *avatars*—photos of unique popular figures—to represent different team members. Much like the tasks, the team members get to choose their own avatars. Such observations supplemented the data derived from interviews and strengthened the understanding of these practices. Figure 5.3 shows the use of avatars to self-assign tasks. A closeup of a task in the figure a task written on a post-it; the estimated effort involved in the task represented in hours: "*5hrs*"; and the owner of the task represented by their avatar. An observation of a New Zealand team's story board is presented below:

Observation of a team's story board, New Zealand:

> "*I looked at the different charts around the room. The Scrum Master (played by a Project Manager) explained to me the convention used for estimating the tasks. The highest priority tasks were labeled '1', then the next highest was labeled '2' and the last was '3'. Similarly the story was designated points and these were 4 digit numbers, the 1st digit was the priority (business value). Priority 1 meant that story was a 'must-have', while 2 was a 'should-have' and 3 was 'nice-to-have' (depending on 1 and 2 being completed in time). The rest of the digits were the estimation for the task. The Scrum Master said that while 2 digits*

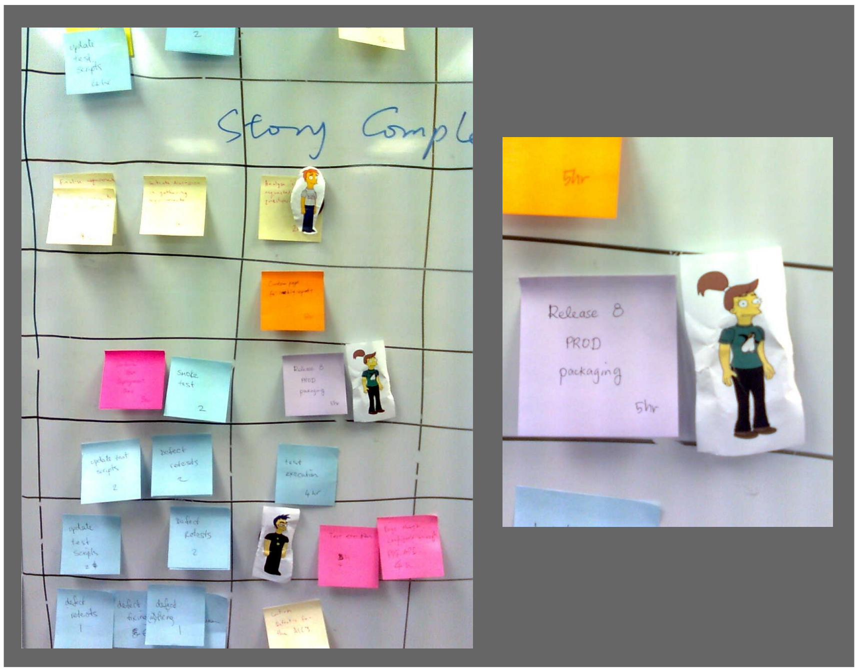


Figure 5.3: Use of avatars to self-assign tasks

> *would be enough to estimate the stories, the third digit helped to space them well on the spreadsheet that he maintained. The charts were divided into 3 columns of 'not checked-out' (not assigned), 'check-out' (assigned to someone), and 'done' (completed). The one common tester for the teams would run the tests (integration, regression, etc) once a task was complete or 'done'.*"

Individuals display responsibility in using their freedom to self-assign by picking the tasks in order of business priority as defined by their customer representatives, instead of picking tasks that are technically more appealing to them.

> "*So focus is on delivering business value as soon as possible - as a result of that you take items which are most required from point of view of business.*" — P27, Developer, India

In situations where several tasks are of the same business value or priority, individuals display their responsibility towards other team members and avoid picking tasks based on ease of implementation. The high level of transparency provided by the story board reinforces the need to pick tasks responsibly:

> "*You're assigning to yourself but you're part of this team of people so you know that people aren't stupid...we joke about choosing a particular thing and we laugh about them being easy or not.*" — P11, Developer, New Zealand

> "*Individuals sign up for easy stories [is] visible, [there is more] sense of responsibility*" — P40, Senior Management, India

Individuals try to avoid potential conflicts during self-assignment. For instance, members in some teams unofficially announce the task as they pick it from the wall such that any potential conflict is easily raised by others and mutually resolved. Such actions display responsibility towards other team members when using the individual freedom to self-assign tasks.

For most individuals, self-assignment leads to taking task ownership. Some individuals, however, struggle to take ownership of tasks during the initial stages of becoming a self-organizing Agile team. Initially, this problem seemed to be related to the Indian hierarchical culture where managers are expected to make all decisions, however, some individuals in New Zealand teams also showed the same resistance to ownership and responsibility [4, 15, 150, 161].

> "*It takes time for people to get out of that mind set that some body is going to be assigning me tasks; coming out of that model of delegation...here [it is] more about taking ownership*" — P27, Developer, India

This initial struggle to accept freedom and use it with responsibility is not based on national cultural differences, rather it is a result of the lack of experience of working in an Agile environment. Using the freedom available in an Agile environment with responsibility requires "*people to be pro-active and do things for themselves*" (P34) and "*assign[ing] to themselves needs maturity*" (P39). Relatively inexperienced Agile teams have issues with accepting autonomy and keep looking up to their seniors and Agile coaches for guidance and decision making. More mature Agile teams, however, are able to effectively balance the freedom to self-assign using story boards with the responsibility to take task ownership.

5.1.3 Self-Monitoring

In order to ensure that they achieve their goals, self-organizing Agile teams carefully monitor their progress through an iteration. Generally, in non-Agile teams, monitoring overall team progress is a responsibility of the team manager. In self-organizing Agile teams, however, this responsibility is shared collectively among all the members of the team.

The team participates in **daily standup** meetings, that allow them to gather a complete view of the team's overall status. Each member of the team provides a quick update on what they achieved the day before, what they are planning to do today, and the impediments they are facing, if any. The daily standup is a simple, yet effective, way of keeping all members of the team abreast of the others' progress, and therefore the progress of the team as a whole. The daily standup also facilitates the surfacing of impediments faced by individual members so that they can be discussed as a group or escalated to senior management for resolution. Observations of daily standup meetings were made for a distributed Indian team where one distant member of the team participated through video-conferencing and for a co-located New Zealand team.

Observation of a team daily standup, New Zealand:

> *One of the teams got together for their daily standup—three developers and one Scrum Master (project manager). The mem-*

bers discussed what they had achieved the previous day and then discussed what they planned to do today. They were also able to spot and resolve dependencies simply by informing each other of their daily progress. The atmosphere was relaxed and professional. There were no tangents, and only relevant issues were discussed. The meeting lasted for about 10 mins. The Scrum Master supplied information such as contacts for resolving certain issues. Some technical details were also covered. Then the team went back to work.

The daily standup serves as an important self-monitoring tool. Team members inform each other about their daily progress. A lack of progress is immediately visible during a meeting and brings on peer-pressure to deliver.

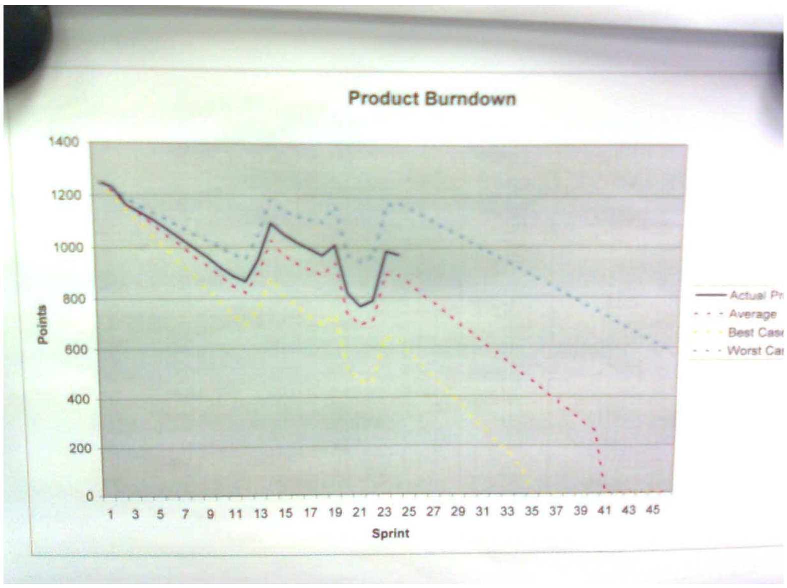


Figure 5.4: Burndown chart tracing the actual (solid black line), average (dashed red line), best (dashed yellow line), and worst (dashed blue line) burndown rates for the team over several sprints

"you lose your comfort zone; you like to finish your test before the new daily meeting. Because going every day and saying 'oh I

didn't finish yet' is terrible; you cannot do that for one week!...so it pressures you to do something and it's not your boss pressuring you, [it is] peer pressure!" — P14, Developer, New Zealand

Self-organizing Agile teams keep a track of their progress through the use of **information radiators**—artifacts that radiate project information with ease and high visibility. **Story board** is a means to track individual and team progress as well as a tool for self-assignment (discussed in section 5.1.2). "*When someone looks to the board they can see who is working on which task*" and "*everyone can read their sticks and see what should be done*" (P14).

Another information radiator is a **burndown chart**—a graph that traces the number of complexity points remaining versus the number of iterations, also called the burndown rate. Usually, the burndown chart will also feature the ideal burndown rate needed to achieve the iteration goal. A quick look at the two rates, ideal and real, traced on the same graph, informs the team of their progress. The graph is drawn or printed by the team on paper, updated regularly, and placed in a visible area as a ready reminder.

Figure 5.4 shows a picture of a product burndown chart, taken during observations of the workplace of one of the participating teams. This product burndown chart traces the progress of the team over the entire length of the project. The complexity points are represented on the vertical axis and the number of sprints (iterations) are represented on the horizontal axis. The solid black line traces the teams' actual burndown rate; the dashed red line shows the teams' average burndown rate; the dashed yellow line indicates the teams' best burndown rate; and the dashed blue line shows the teams' worst burndown rate. This burndown chart shows that the team is currently in the 24th sprint and is performing slightly worse than their average rate, but much better than their worst case.

Daily standup meetings and information radiators such as story boards and burndown charts allow a self-organizing team to monitor their progress through an iteration and remain on track with achieving their own team goal.

When Agile teams are provided freedom by the management to organize and manage their own affairs, it fosters "*self-monitoring, self management, higher levels of commitments and responsibility*" (P36). Team members are found to be "*putting their hand up to do stuff*", and they "*get better [at] organization*", and at the same time there is "*a lot more ownership*" and "*sense of responsibility and accountability*" (P34, P40). Self-organizing Agile teams are aware of their responsibility to adhere to Agile practices, responsibility towards each other, and responsibility to achieve team goals.

Teams enjoy freedom both at an individual level as well as a team level. Similarly, they display responsibility both at the individual level and at the team level. Through the practices of collective decision making (collective estimation, planning, collectively deciding team norms and principles, and self-committing to team goals), self-assignment (using story boards while taking task ownership), and self-monitoring team progress (through status report meetings and information radiators), self-organizing Agile teams balance between the freedom to commit to their own goals and the responsibility to achieve those goals.

5.1.4 Consequence of Imbalance

The importance of balancing freedom and responsibility is most apparent when a team is unable to use their freedom in a responsible manner. For example, the general manager of an Agile organization in India shared an experience where they had to intervene with a self-organizing team which was unable to balance successfully between freedom and responsibility (P53). The team had a couple of senior developers who were extremely proficient at their tasks, but were misusing the freedom provided and were dictating and overriding the rest of the team. Their influence had become so strong that it led to a clear divide in the team between those that sided with them in every decision fearing comeback and the few that still tried to be democratic. These members had clearly lost their sense of responsibility towards other team members by not including them in decision making. The Agile Coach, acting as a *Terminator*, sought senior management interference and removed those senior developers from the team. The rest of the team took some time to return to their previous self-organizing state. The consequence of imbalance between freedom and responsibility is generally senior management intervention, restricting the team's ability to self-organize in the short term. Agile teams attempt to carefully balance freedom and responsibility in order to avoid senior management intervention and sustain their self-organizing nature.

5.2 Balancing Cross-functionality and Specialization

A defining characteristic of self-organizing Agile teams is their ability to maintain cross-functionality in the team. Cross-functionality is the ability of team members to (a) look beyond their organizational roles (such as developers, testers, and designers, etc) and to take an interest in activities outside their areas of specialization, and (b) to look beyond their technical areas of expertise (such as database management or graphical user interface (GUI) design) and take the opportunity to expand their expertise in other technical areas. Cross-functionality allows team members to gain a more rounded vision of the project through understanding it from multiple perspectives.

While Agile teams generally promote cross-functionality, they cannot completely dispose of specialization. Some amount of specialization is needed in both functional roles and in the technical areas of expertise. Self-organizing Agile teams perform a balancing act between encouraging cross-functionality and accepting the need for specialization. The practices that enable this balancing act are described in detail below, along with an example of the consequence of imbalance.

5.2.1 Need for Specialization

Self-organizing Agile teams are comprised of individuals with diverse abilities and perspectives. The presence of **multiple perspectives** on the team provides individuals with opportunities to share and learn from each other, in other words become more cross-functional. For example, developers help with testing when needed, and testers try to understand the developers' perspective. Team members with different organizational roles interact and collaborate with each other in order to gain better understanding of each other's functional perspectives in the larger scheme of the project.

> "*The whole thing with Agile is getting people to be more cross-disciplinary, to take an interest in somebody else's perspective, to stop this artificial division between developers and analysts and testers.*" — P20, Agile Coach, New Zealand

The boundaries created by their formal organizational roles are blurred in self-organizing Agile teams, as individuals learn multiple perspectives and become more cross-functional. Multiple perspectives on the team leads to a learning environment.

> "*[In Agile teams] no egos, no belief that their technical solution ideas are better than anybody else's...In other [non-Agile] teams, someone will have an idea and as soon as someone else has an counterbalance idea then they fight over the idea and that doesn't seem to happen with Agile.*" — P18, Senior Management, New Zealand

Teams carefully balance between encouraging cross-functionality and recognizing the need for specialization. For example, developers and testers perform their specialized tasks first, before helping out with the others' tasks within the limitations of their cross-functional abilities.

> "*You choose anything that you wanted, generally testers would stick to testing first, BAs would stick to requirements first, and developers stick to development first...[but] as we progress, obviously a lot of the BA work dies down so I'll say...'can I help with development?' And someone will say 'well this bit's quite easy'...so I'll go in and just assign [it to] myself.*" — P4, Business Analyst, New Zealand

Venturing outside the areas of technical expertise was not always easy. Some individuals are uncomfortable in practicing cross-functionality because "*now they [team members] are switching role...people don't want to come into different shoes, different hats very frequently*" (P39). The fear of exposing inadequacies when attempting a task outside the individual's area of expertise leads individuals to specialize more often than become cross-functional.

> "*Sometimes I'm afraid because I don't know how to do that story, and at that point I make a decision, I take a risk - what is the risk? Oh, I have to expose myself as ignorant in that subject! And sometimes it's easier if you just take a task that you know how to do and you just do it quickly and complete it. Sometimes I take the risk, sometimes I don't.*" — P14, Developer, New Zealand

A culture of collaboration and cooperation in an Agile team is crucial for team members to overcome such apprehensions and explore other areas of expertise. Most mature self-organizing Agile teams are highly cohesive and cooperative, helping each other learn new skills across different technical areas.

> "*We just didn't do things based on technical skills...people would just grab whatever and if they couldn't do it themselves, they get help. And that worked well.*" — P11, Developer, New Zealand

Understanding each other's perspectives implies individuals can potentially step into each others' roles in the face of unforeseen loss or unavailability of individuals performing within specialized organizational roles:

"*recently our tester left the project...we [developers] needed to step up and do some testing ourselves*" - P13, Developer, New Zealand

"*When we are short of testing capacity in the team...even I have done some testing for a fellow developer on a user story, which is pretty normal.* – P31, Agile Coach, India

Self-organizing Agile teams recognize certain limitations to cross-functionality. For example, even though developers try and understand the testers' perspective, they can harbour inherent biases towards their own code which prevents them from recognizing weaknesses in it.

"*If all the developers can think in the way a tester thinks then I think we [testers] are not required! (laughs) But actually that doesn't happen because some sort of biasing is always there for their own code.*" — P32, Tester, India

The presence of multiple perspectives on the team ensures that cross-fertilization can happen. At the same time, the need for some amount of specialization is acknowledged.

5.2.2 Encouraging Cross-Functionality

Self-organizing Agile teams encourage cross-functionality through the practice of **group programming**, where team members work together in an open-plan workspace while frequently collaborating with each other. Open-plan workspaces have no cubicles and all team members and the project artifacts are highly visible. Figure 5.5 shows a picture of an open-plan workspace environment of one of the Indian self-organizing Agile teams participating in the research. The term group programming emerged from data analysis, and is an example of an *in-situ* code—a code derived directly from the participant's comments.

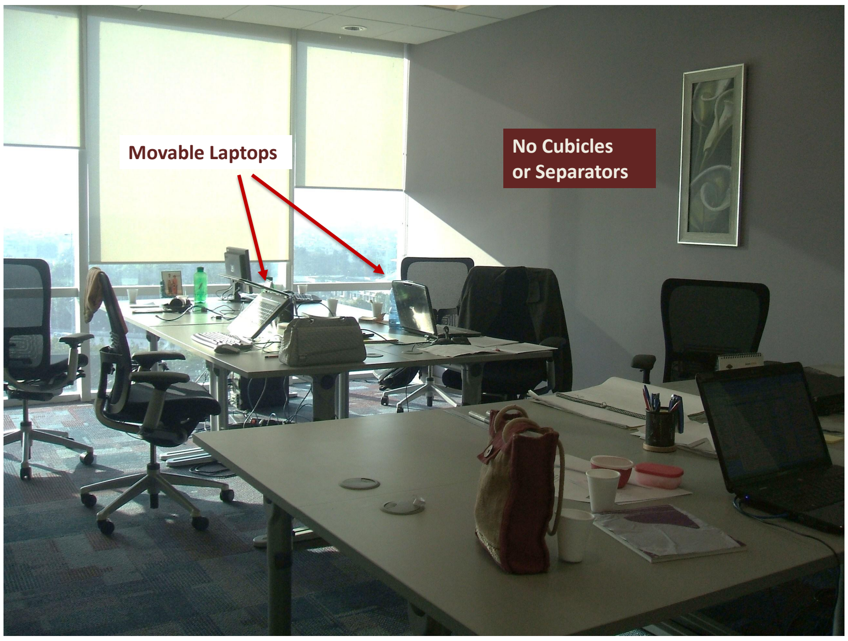


Figure 5.5: Open-plan workspaces enable *Group Programming*

> *"I think Agile software development is not good, or not an ideal one, for people who love to programme all alone, because some of the developers prefer working all alone and concentrating on their stuff and nobody can disturb them, but Agile is totally different from that, so it's sort of* ***group programming****...doing it all alone...would have been quite a difficult job, but...all [of us] were involved in all the tasks and everything"* — P16, Developer, New Zealand

A main consequence of group programming is that it puts developers and testers together in the same physical space. A result is that instead of being pitched against each other in separate development and testing teams, developers and testers work together on one team. Developers value the testers' perspectives and often seek their advice when implementing functionalities in code. Similarly, testers often engage in discussions with the developers in a bid to understand the decisions behind their implementations.

> "*If I think I'm writing something that is a bit tricky then I pull the tester over to sit with him and say...this is how it's looking, because they tend to have a different view on things and sometimes as a developer you forget the other view and you need to step back and get that input. So I quite like to...get them involved.*" — P13, Developer, NZ

> "*The developers getting used to actually having to treat the testers with respect, and the testers acknowledging that um, the developers might actually have some good ideas occasionally (laughs)*" — P26, Agile Coach, NZ

Direct communication between developers and testers not only saves time and effort, but also promotes cohesiveness in the team.

> "*We'll [developers and tester] be having a root cause analysis and we sit together and see this is the problem—why this was not implemented or if the developer has misunderstood that, then we sit together. There is nothing [like] that we're going to product owner and telling him that your stuff was not done then he is telling the developer why didn't you do that...unnecessarily there's a loop.*" — P32, Tester, India

Group programming promotes a collaborative team environment that is particularly useful for newcomers on the team.

> "*...[when] you start doing it [a task] and you face some of the problems...okay, so this is a bit difficult now, I never thought of this thing...you can surely go to another developer, because he's also knowing about that task because he was present there at the time of breakdown...so he can immediately help you...[or] you can*

> *leave it in between and just tell the other developer 'okay, I'm having a problem so can you please look at that [and] I can pick up some other task'.*" — P16, Developer, New Zealand

A newcomer with a traditional software development background may easily feel overwhelmed in a new Agile environment. Group programming allows for a cohesive, learning environment where newcomers find support from their team-mates.

> "*The day I joined...They [team] held me because I was not able to move. Because what I feel is when you join a new organization and that too from a traditional to a new Agile methodology you have to have some space for yourself, some room. But they [team] hold my finger and they didn't ask me to just walk—they let me run with them! And that was the best thing that I have seen and I really appreciate that part of the developers that...they helped me a lot.*" — P32, Tester, India

Cross-functionality in self-organizing Agile teams is not limited to crossing the boundaries of organizational roles. Cross-functionality also includes the teams' ability to actively seek opportunities to work outside their areas of technical expertise (within the same organizational role.) For example, developers specialize in different technical areas of expertise such as database management or graphical user interface (GUI) design. In self-organizing Agile teams, developers try to work outside their specialization areas and actively embrace opportunities to gain expertise in other technical areas. One of the advantages of this practice is that team members become familiar with most technical aspects of the project so they can easily manage any area, which is consistent with XP's collective code ownership principle [19]. The practice of group programming supported collective code ownership among team members.

> "*So we encourage people not to get boxed into 'I only do database access stuff!'...One of our keys is that we want everyone to know*

> *as much of the code base as possible, so that if someone leaves or can't work on another problem because they're busy, someone else should be able to come in and at least feel a little bit familiar with what's going on*" - P13, Developer, NZ

> "*From the sprint backlog you want to pick the XML parser task or you want to pick the GUI design task that is entirely up to you and that is the freedom that Agile gives you.*" — P29, Developer, India

Flexibility to work in multiple technical areas is welcomed by developers because it helps them maintain interest in their work. As one of the developers nearing the end of their Agile project noted:

> "*I think the thing that I will probably miss the most, in Agile, is the fact that everything is so flexible; that one day you can be doing one thing and the next day you can be doing something else.*" - P2, Developer, NZ

Another practice that promotes cross-functionality across teams is, **rotation**. Rotation is a policy that is used across large projects with multiple teams, and allows individuals to rotate across the teams, giving them a wide exposure to a large number of different contexts, development platforms, languages, and technical areas of expertise.

> "*We rotate across teams...so that's one good way of building knowledge in the system...so particularly people who are less than a year out or so—they don't know Ruby so I mean that's fine...you can get on and, you know, learn the technology, learn all the skills.*" — P46, Developer, India

The rotation policy is based on the premise that face-to-face communication and collaboration leads to better transfer of knowledge and skills.

Rotation allows individuals to learn the tricks of the trade through hands-on experience with people from different backgrounds. In an Indian Agile organization, rotation was used as a strategy for knowledge sharing.

> "*[speaking about rotation policy]...it is part of our beliefs that face to face transfer of knowledge, that's the best way to do it, yeah we have all those virtual tools in place, we have mailing lists...and the rest of it, but I think the face to face, you know, hard-back, really helps in knowledge sharing and collaboration, that's the fundamental reason [behind rotation]. And it also helps in the transfer of all kinds of engineering practices, you know, tips and tricks that people run on the ground, that gets shared.*" — P51, Knowledge Strategist, India

Rotation is also used as a means to keep the work environment interesting for the team members by exposing them to changing and challenging new areas (P20, P34, P46, P51).

5.2.3 Consequence of Imbalance

The need for balancing between cross-functionality and specialization when working in a self-organizing team is highlighted by the example below. A business analyst (BA) on a New Zealand team misused their cross-functional programming skills in secret and had started causing damage to the project code base. The BA would use their coding skills to work on the code base without the knowledge of the developers and causes lots of confusion and errors in the system. In this case of irresponsible cross-functionality, the BA's unofficial involvement in programming caused the team several hours of rework. The Agile Coach on the team took on a *Terminator* role [78]—securing senior management support to remove the business analyst. The team performance rose dramatically afterwards, as confirmed by their customer representative:

> "*[When] we got our scrum coach in...that BA was moved to another project and their contract was not renewed... Once we had [the coach's] involvement the work got back on track—we'd gone four months down the wrong road and [the team] were able to get us back to where we should be in, I think, about six weeks.*" — P9, Customer Representative, New Zealand

A team's failure to balance between cross-functionality and specialization, as in this example, invites senior management intervention. Frequent senior management interference poses a threat to the team's self-organizing nature. Self-organizing Agile teams carefully balance between cross-functionality and specialization in order to avoid senior management intervention and sustain their self-organizing nature.

5.3 Balancing Continuous Learning and Iteration Pressure

Software development teams need to keep themselves abreast of the latest technologies, processes, and tools in order to manage dynamic requirements and market trends. Continuous learning is all the more important for self-organizing Agile teams because *responding to change* is an essential principle of Agile software development [19].

> "*I think in our business, software developing, it's a complex subject and it's impossible for one person to know about everything, so it's a day-by-day thing...This is a normal step and everybody is learning each day.*" — . P14, Developer, New Zealand

Self-organizing Agile teams recognize the need to indulge in continuous improvement powered by constant self-evaluation and continuous learning:

> "*I think we just need to keep going and we need to keep improving. I think the minute you think you're there, you're not. Because you*

> *can always do better, you can always learn from what went well, what didn't go well and tweak things slightly.*" — P13, Developer, New Zealand

Continuous learning involves different types of learning — learning Agile practices, learning new or complex technical skills, learning cross-functional skills, and learning from the team's own experiences — all of which fuel self-improvement. The rest of the section describes how Agile teams perform a balancing act between continuous learning of different types and the pressure to deliver the team goal every iteration. An example of the consequence of imbalance is also presented.

5.3.1 Self-Evaluation

Self-organizing Agile teams perform self-evaluation through the practice of **retrospectives**. Retrospective meetings are held at the end of each iteration where the team collectively self-evaluates themselves by addressing four aspects: what went well, what didn't go well, bouquets, and suggestions for improvement [48].

> "*I think that sort of fits in well with the whole idea of Agile, where you're constantly going 'is this working for us as a team? or for me as an individual?'* — P13, Developer, New Zealand

Retrospectives are used as an effective tool to evaluate the learning by the team over an iteration and suggest concrete steps for improvement.

> "*With every retrospective we certainly came up with ideas to improve our process, and I think with all those retrospective sessions under our belt, with all the experience sizing, planning, everything combined, it really made us evolve as a team. I'd certainly say our team dynamics expanded well beyond what we thought they would. At the moment we're exceptional, we're just a little family that works together.*" — P4, Business Analyst, New Zealand

Retrospectives are a powerful mechanism for the team to engage in self-evaluation and self-correction:

> "*The key here that makes it all work is this practice of retrospectives. Because that essentially says you say stop, 'how are we doing guys? What are the good things that we're doing, what are the not so clever things that we're doing, how do we stop the not so clever things, how do we start better things?' Because then with this practice and with the continuous kneading out the things that don't quite work and focusing on the things that work, you grow that eco-system, you develop it, and you're bound to be successful.*" — P20, Agile Coach, New Zealand

Along with the need for continuous learning and improvement, Agile teams are very much aware of the pressures of delivering their iteration goals. Agile teams face **iteration pressure**—the pressure to deliver to a committed team goal every iteration. Iteration pressure, in itself, is not detrimental to the team, in fact some amount of iteration pressure is necessary to motivate teams to deliver their goals. Short iteration lengths or an extremely high and unsustainable development velocity, on the other hand, can cause excessive iteration pressure. For instance, a developer found one week iterations to be very demanding:

> "*I'm always feeling the need to rush, rush, rush!...after one week [iteration], we want to remove all these stickies [tasks] from the wall. So it's always pressure...if you have [longer] development time, then I can adjust my work like if we spent a little bit longer than we expected, I can catch up next week.*" — P15, Developer, New Zealand

Creating and maintaining a continuous learning environment requires teams to set some explicit time aside for learning each iteration. Iteration pressure, on the other hand, implies they may not have any extra time to spare:

> "*You need to actually allow time for other team members to learn what you do and for you to learn what they do. Often we tend to fill up our sprints with so much that a good teaching environment isn't necessarily there...they can see what you're doing but you need to be able to take the time to explain in really good detail.*"
> — P8, Tester, New Zealand

Retrospectives can be used to assess whether the iteration pressure is unbearable for the team and suggest ways to overcome it. During an interview, a tester revealed that they were facing iteration pressure because "*testing was always pinched at the end*" and resolved to take the matter up in a next retrospective because "*that's what [retrospectives] are for*" (P8). Participants found retrospectives to be "*a key ingredient in Agile methodology*" (P20) which allowed them to evaluate team practices, including team velocity, and correcting them as needed.

5.3.2 Self-Improvement

Team members have the desire to learn new and better ways of working but are sometimes too pressured by the iteration tasks to be able to devote any time to learning and improvement:

> "*I'd be interested to learn various Agile techniques for requirements gathering, such as events and themes, and I'd love to try and use some of them in an Agile project. It's just [that] I haven't really had a lot of time to think about it. [Scrum] is very action oriented.*" — P4, Business Analyst, New Zealand

A practice that allows self-organizing teams to allow for learning while managing iteration pressure is a **learning spike**. A learning spike is an exclusive time set aside—within an iteration or spread across multiple iterations—for learning. After performing self-evaluation through retrospectives, the team may discover that they are lagging behind in a particular area and decide to devote some exclusive time to update themselves in that area.

One New Zealand team faced excessive iteration pressure when their only tester on the team left unexpectedly. The team realized the need to automate their testing efforts. The Agile Coach helped the team not succumb to the iteration pressure and the team created a learning spike to improving their testing. The improvement involved the team learning new tools and techniques and implementing their own automating testing framework.

> "*We've just basically reduced our velocity and taken the time to do those things because we knew they were important. We made a call that we were going to not going to wimp out, and go back to the manual testing...make it automated...the new tester had more coding skills and therefore we've taken automation a lot further. ...we seem to be the only team I can find in New Zealand doing one hundred percent automation.*" — P17, Agile Coach, New Zealand

The whole team may not be involved in the learning spike. While some members perform the learning spike, other can continue to work on regular stories and tasks, thereby managing iteration pressure to an extent:

> "*amongst five of us two of them they started with testing stuff and how to do that and then the three that were left with the development and the other stories. But for a week or two we really...everybody was thinking that what approach should be used for the testing stuff so that time we had to switch some roles from developers to testers and back and forth.*" — P16, Developer, New Zealand

Another source of learning comes from **pair-in-need**, a modification to the standard XP practice where developers work in pairs on every task [19].

Teams practicing primarily Scrum and combinations of Scrum and XP, practice pair-in-need where pairing was done on a need basis, rather than most of the time, to "*distribute knowledge*" (P30) and complete complex tasks (P44).

> "*The way we do it is that if things are unpredictable we always take up user stories as a pair. There are written tasks for which we don't really need to sit together we can part, but if something requires—this is complex, this is design-intensive—we sit together and pair it.*" — P31, Agile Coach, India

Collaboration through pair-in-need becomes an important source of learning. Agile coaches in relatively new teams and senior team members in mature teams often take on a *Mentor* role to help newcomers learn the basic Agile practices and catch up to the team's velocity [78].

> "*I had never worked on the Spring framework before, but in this project it's completely related to Spring framework, and Spring transaction management and all, so I started learning it...we were pairing each with other, that time it was beneficial because the other person was quite okay...and he knew about the Spring framework and he had done it before in some other project. So it helped me to learn it more faster, because he used to say: 'okay, you have to go with this stuff, and you can do it'. So that was a major advantage.*" — P16, Developer, New Zealand.

In order to balance continuous learning and iteration pressure, helping team members through collaboration should be considered acceptable by the team as a task that promotes both learning and delivering the iteration goal:

> "*[We] help [each other], so that means that the next day's stand-up you knew that you were helping...so that's all right...because I'm covering someone.*" — P11, Developer, NZ

Pair-in-need works well for these teams because it allows them to learn how to tackle new and complex tasks with the help of a peer and at the same time to move closer to their iteration goal.

5.3.3 Consequence of Imbalance

Teams must balance learning and iteration pressure. A developer shared an experience where their team had committed to too much in an iteration, thereby bringing excessive iteration pressure upon themselves. The team was unable to keep up with the self-imposed high velocity which resulted in tests failing across the board:

> "*We'd gotten a bit over-confident and we'd committed to too much in the sprint... Everyone was feeling like 'we have to get through [all the tasks]'...then I started testing and everything fell over!*" – P25, Developer, NZ

In the following retrospective, the team decided to take a step back and put some guidelines in place regarding their velocity. They decided to focus on quality and not just quantity of the tasks. The team also decided to learn and set up better guidelines for testing.

> "*So we looked at that retrospective and thought 'okay, that was a complete [mess], how can we make sure it's not next time?'...We decided that...what we delivered had to be working, and that meant that if it took longer and if some of the stuff had to be dropped until the next sprint then that's what happened!*" – P25, Developer, NZ

A balance between continuous learning and iteration pressure is necessary to allow Agile teams to keep improving and transcending beyond their current abilities.

5.4 An Integrated Set of Practices

The three balancing acts are highly inter-related and re-enforce each other in several ways. The balancing acts include several low-level practices that enable self-organization on an every day basis. Balancing freedom and responsibility involves practices such as collective decision making through collective estimation and planning, collectively deciding teams and principles, and self-committing to team goals; self-assignment using story boards; self-monitoring through daily standup meetings and use of information radiators. Balancing cross-functionality and specialization involves practices such as multiple perspectives, group programming, rotation. Balancing continuous learning and iteration pressure involves practices such as retrospectives, learning spike, and pair-in-need. These practices are closely related to each other and support each other. Figure 5.6 depicts the relationships between the different self-organizing Agile team practices. This integrated set of practices specifically facilitates self-organization in Agile teams. The relationships between the practices are described below. There are other Agile practices that teams engage in. These include XP practices such as metaphor, refactoring, etc. While these practices enable proper functioning of a development team, they do not specifically facilitate self-organization and are not discussed here.

In self-organizing Agile teams, the whole team is able to participate and contribute to collective estimation and planning. As developers, testers, designers, business analysts, etc all collectively estimate and plan their iterations, it fosters a good understanding of the project from multiple perspectives. Collective estimation and planning also promotes group programming as team members share common ideas about the stories and tasks at a high level which they later program as a group. Team members can indicate their interests in selecting certain tasks during the estimation and planning session as a pre-cursor to self-assignment of those tasks later from the story board. Since the estimation and planning is done collectively, team members have an understanding of the efforts involved in various tasks. This leads to transparency about tasks estimates, which in turn promotes responsibility

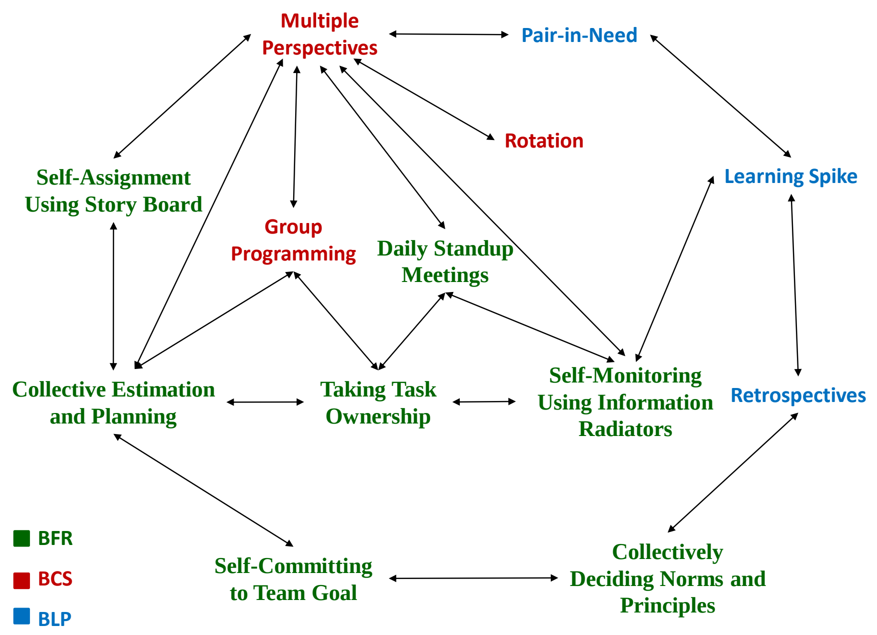

Multiple Perspectives
Pair-in-Need
Rotation
Learning Spike
Self-Assignment Using Story Board
Group Programming
Daily Standup Meetings
Collective Estimation and Planning
Taking Task Ownership
Self-Monitoring Using Information Radiators
Retrospectives
Self-Committing to Team Goal
Collectively Deciding Norms and Principles
🟩 BFR
🟥 BCS
🟦 BLP

Figure 5.6: Self-Organizing Agile Team Practices Support Each Other (Balancing Freedom & Responsibility (BFR); Balancing Cross-Functionality & Specialization (BCS); Balancing Continuous Learning & Iteration Pressure (BLP)

among team members to not only complete the tasks they choose, but also finish them within the estimated time. This transparency is further enforced through daily standup meetings and the use of information radiators.

Self-assignment involves team members picking tasks to perform instead of being delegated tasks. A key motivation during self-assignment is to select tasks with the highest business value and not necessarily tasks that are easy to perform. As a result, team members often pick up tasks that are of high business priority but are well outside their area of expertise, which provides an opportunity to gain new cross-functional skills. Thus, the practice of self-assignment promotes cross-functionality in self-organizing Agile teams.

The daily standup meetings allow team members to understand the project from a range of viewpoints which promotes multiple perspectives on the team. Information radiators, such as story boards and burndown charts, are primarily tools for self-monitoring progress. These information radiators foster responsibility among team members to complete selected tasks. However, information radiators also highlight the areas in which the team is not performing optimally. For example, a large number of testing tasks stagnating on the story board is an indicator of either poor quality code being produced or that the tester (often outnumbered by developers) is unable to manage the testing load. This presents an opportunity for self-improvement where the team may decide to improve their quality of code or automate some of their testing (section 5.3). It also presents an opportunity for cross-functionality where developers may pitch in to help the tester with testing tasks.

The practices of group programming presents a conducive environment for sharing multiple perspectives and encouraging cross-functionality. Other practices that promote cross-functionality are rotation and pair-in-need. While pair-in-need primarily enables the team to balance continuous learning and iteration pressure, it is also an important means for encouraging cross-functionality.

As well as self-monitoring daily via standup meetings and information radiators, self-organizing Agile teams also perform self-evaluation on an iteration-by-iteration basis through the practice of retrospectives. Retrospectives present an opportunity to evaluate a team's ability to perform all three balancing acts. As a result of this self-evaluation, a team may decide to concentrate on self-improvement in several areas, such as create a learning spike to resolve an immediate need. A team may also decide to re-evaluate and adapt their norms and principles, such as team velocity, defect tolerance, work hours etc.

The practices of self-organizing Agile teams described in this chapter, support and complement each other much like the XP practices (section 2.2.2) support each other.

The three balancing acts enable the team to balance short term gains with long term benefits. For example, self-assignment of tasks provides an opportunity for immediate gains to individuals in terms of freedom to choose whatever task they want from the story board. However, team members choose tasks based on high business value, displaying responsibility towards customers; and based on an awareness of other members' preferences or interest, displaying responsibility towards the team. An inability to balance freedom and responsibility invites senior management intervention which takes away their long term ability to self-assign tasks (section 5.1).

Similarly, confining themselves to their specialized areas of expertise allows team members to achieve faster results in the short term (for example, within an iteration). However, acquiring cross-functional skills enables them to reap long term benefits of achieving sustained progress by removing functional dependencies on individuals (for example, throughout a project or through multiple projects).

Finally, achieving a high team velocity in an iteration allows the team to attain short term gains. Managing iteration pressure and allowing time for learning and growth, on the other hand, enables long term benefits of sustained velocity. All together the practices enable Agile software development teams to achieve and sustain self-organization on an everyday as well as a long term basis.

5.5 Balancing Acts and the General Principles of Self-Organization

The general principles and specific conditions of self-organization have been explored in Agile literature (described in section 2.3). The concrete practices of self-organizing Agile teams, however, have not yet been established from industry-based research across multiple teams, organizations, and countries. A contribution of this research is the description of concrete, everyday practices that facilitate self-organization in Agile teams and how these practices fulfill the conditions and principles of self-organization.

While most of these low-level practices that make up the balancing acts

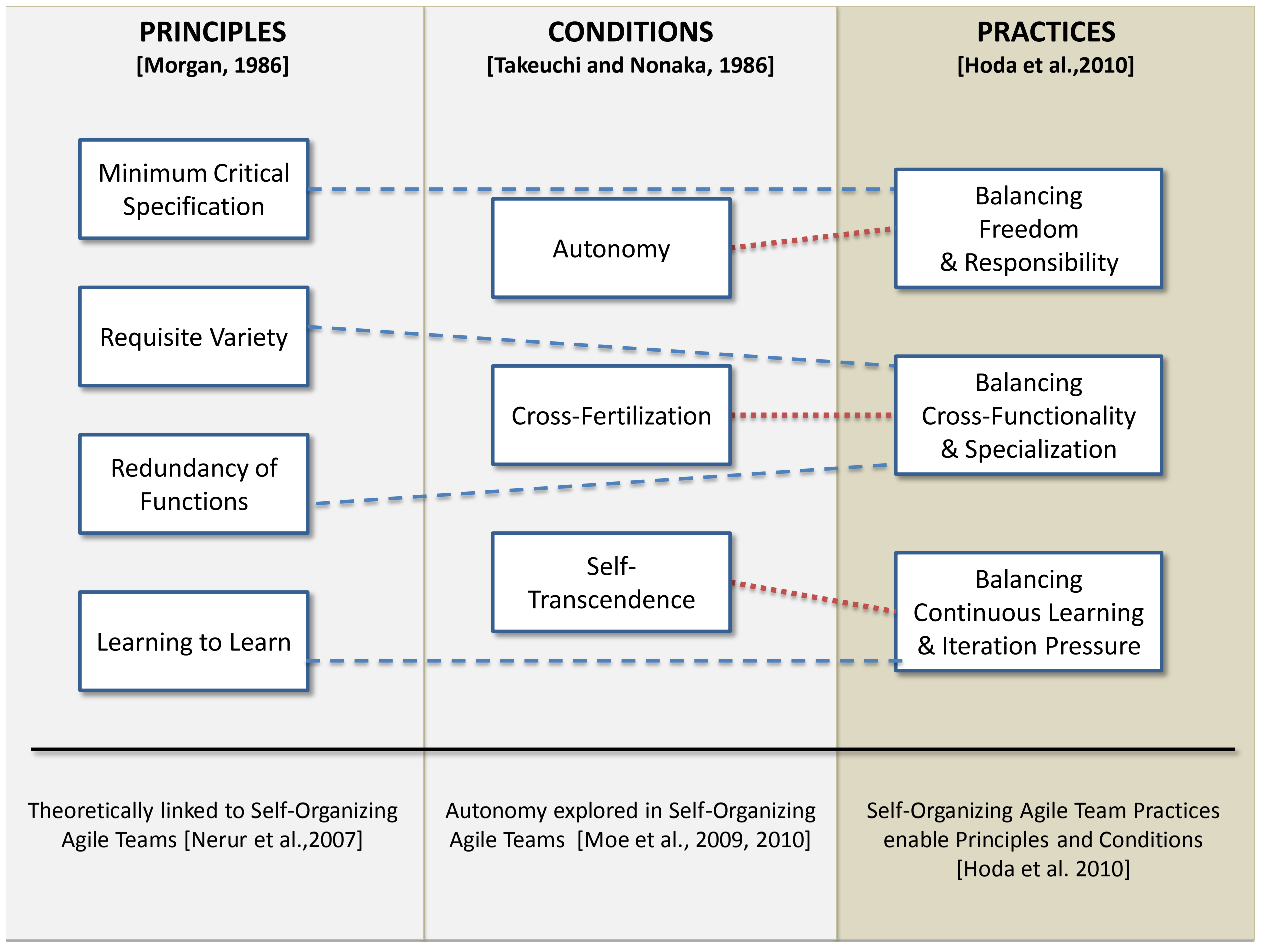

PRINCIPLES
[Morgan, 1986]
CONDITIONS
[Takeuchi and Nonaka, 1986]
PRACTICES
[Hoda et al.,2010]
Minimum Critical Specification
Requisite Variety
Redundancy of Functions
Learning to Learn
Autonomy
Cross-Fertilization
Self-Transcendence
Balancing Freedom & Responsibility
Balancing Cross-Functionality & Specialization
Balancing Continuous Learning & Iteration Pressure
Theoretically linked to Self-Organizing Agile Teams [Nerur et al.,2007]
Autonomy explored in Self-Organizing Agile Teams [Moe et al., 2009, 2010]
Self-Organizing Agile Team Practices enable Principles and Conditions [Hoda et al. 2010]

Figure 5.7: Relationships between the practices of self-organizing Agile teams (the Balancing Acts) [76] and the general principles of self-organization (Morgan, 1986) [115]; and the Balancing Acts and the fundamental conditions of self-organization (Takeuchi and Nonaka, 1986) [154] (indicated by blue and red dotted lines respectively.)

are standard or adapted practices from Scrum and XP, they specifically enable self-organization in Agile teams. The following sections describe how these practices relate to the general principles of self-organization from an organizational perspective and to the fundamental conditions of self-organization as applied to Agile software development. Figure 5.7 depicts the relationships between the balancing acts, the specific conditions of self-organizing Agile teams [154], and the general principles of self-organization from a organizational perspective [115].

The four principles of self-organization described from an organizational perspective are: minimum critical specification, requisite variety, redundancy of functions, and learning to learn [14, 115]. Several researchers have studied and used some or all these principles to explain their findings or further their research [86, 113, 114, 118, 120]. The relationship between these principles and the balancing acts are discussed below.

Minimum Critical Specification

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 [115]. Morgan also emphasizes the need for self-organizing teams to work in an environment of "*bounded*" or "*responsible autonomy*" [115]. Hut et al. [86] note that the role of management is extremely important in providing autonomy to the team and for team empowerment. Our theory confirms that freedom provided by senior management is extremely important for Agile teams to self-organize. Hut et al. [86] suggest that while interventions by senior management can "*dramatically undermine empowerment*", such interventions "*may sometimes be inevitable*". As such, they propose "*boundary management*" in order to find the "*right balance*" [86]. Our research found that senior management was forced to intervene at times when the teams crossed their boundaries of freedom, in an effort to restore the balance. Similarly, Mollenman [114] discusses the need for "*balance of power*" which is described as the balancing act between freedom and responsibility, in our theory.

Requisite Variety and Redundancy of Functions

Requisite variety is derived from the "*law of requisite variety*" [14] that claims variety can be handled by variety such that a changing organizational environment is best handled by a group containing people with a variety of skills. Morgan, applying this law to organizational theory, defines requisite variety as the need for any control system to match the complexity and diversity of the environment being controlled [115].

Nerur et al. relate this principle to Agile software development by comparing variety among team members to cross-functionality or interchangeable roles [118]. Requisite variety implies that changes in the environment of the organization is 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 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 influential in improving the quality of the employee's working life [86, 114]. Both these aspects of self-organizing teams are well-suited to Agile methods which focus on responding to change and on the people that enable it [19, 72, 138]. Our research found that teams were facing dynamic environments, in terms of changing customer requirements and technologies, and were composed of individuals possessing variety of skills to respond to these changes, thus fulfilling requisite variety [14].

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 [86]. Nonaka refers to this principle as cross-functionality in a self-organizing team [120]. Our research found that teams promoted cross-functionality across technical areas of expertise as well as across functional roles. Multifunctionality (achieved by cross-training) or cross-functionality has been related to improved team performance [114]. However, limitations to cross-functionality, such as expense of cross-training, have also been acknowledged and imply a need for finding an 'optimal level' of cross-functionality for the team [114]. Our research found that while teams promote cross-functionality, they also acknowledge that some amount of specialization persists. Finding the 'optimal level', therefore, is a balancing act between cross-functionality and specialization that our participants performed.

Learning to Learn

Learning to learn refers to the team's ability to reanalyze problems, reappraise the best work method, and reconsider the required output if necessary [86]. Self-organizing Agile teams are able to iteratively solve problems using 'learning to learn' via double-loop learning [115, 118]. The specific Agile practices that facilitate 'learning to learn' include reflection workshops, standup meetings, pair programming, etc [118]. Our research shows that a couple of these mechanisms of double-loop learning—retrospectives and pair-in-need—particularly enabled teams to balance between continuous learning and iteration pressure.

5.6 Balancing Acts and the Specific Conditions of Self-Organization

This section discusses the relationship between the balancing acts and the fundamental conditions of self-organization specifically applicable to Agile software development [154].

Self-organizing Agile teams are meant to exhibit three conditions: autonomy, cross-fertilization, and self-transcendence [154]. After a careful study of the three conditions of self-organizing teams, a relationship between those conditions and the balancing acts was established. Each of the balancing acts were performed in order to uphold each of the three fundamental conditions of self-organizing teams, namely: balancing freedom and responsibility in order to uphold the condition of autonomy, balancing cross-functionality and specialization in order to uphold the condition of cross-fertilization, and balancing continuous learning and iteration pressure in order to uphold self-transcendence. In unison, the balancing acts were performed by the teams in an effort to uphold their self-organizing nature. These relationships are discussed below.

Autonomy

A team possesses autonomy when (a) they are provided freedom by their senior management to manage and assume responsibility of their own tasks and (b) when there is minimum interference from senior management in the teams' day to day activities [154]. Our participants were provided freedom by senior management to manage their own tasks, which fulfills the first criterion of autonomy. In order to ensure there was minimum interference from senior management—the second criterion of autonomy—the teams assumed responsibility in using that freedom. Thus by balancing between freedom and responsibility they ensured that they were able to not only achieve but also sustain autonomy.

Cross-Fertilization

A team possesses cross-fertilization when (a) it is composed of individual members with varying specializations, thought processes, and behaviour patterns and (b) these individuals interact amongst themselves leading to better understanding of each others' perspectives [154]. Our research shows that Agile teams consist of individual members with varying specializations—developers, testers, business analysts—which fulfills the first criterion for cross-fertilization. In order to ensure that these individuals benefited from understanding each others' perspectives—the second criterion of cross-fertilization

—the teams frequently interact across different functional roles and attempt tasks across different technical areas. Teams find it impossible to completely avoid specialization but try to be as cross-functional as possible. A team's ability to balance specialization and cross-functionality means they can achieve and sustain cross-fertilization.

Self-Transcendence

A team possesses self-transcendence when (a) they establish their own goals and (b) keep on evaluating themselves such that they are able to devise newer and better ways of achieving those goals [154]. Our study found that teams are able to establish their own goals in terms of deciding how much to commit to in an iteration, thus fulfilling the first criterion of self-transcendence. Teams not only establish their own goals but also assume full responsibility to achieve those goals causing pressure to deliver. While some iteration pressure motivates teams to achieve their goals, excessive pressure results in a neglect of learning and improvement. In order to balance between iteration pressure and the need for continuous learning, the teams practice pair-in-need to both complete tasks and to learn from each other in the process. The other technique is to engage in retrospective meetings to self-evaluate and suggest ways of improvement. Teams use retrospectives to find a balance in the amount of time they devote to finishing tasks versus the time they spend specifically on learning new and better ways of working. Thus, by balancing iteration pressure and the need for continuous learning, teams were able to achieve self-transcendence.

Most Agile teams display autonomy where senior management provides them with an environment of freedom and trust. Most Agile teams also value and encourage cross-fertilization while maintaining some amount of specialization. Self-transcendence, however, is the most demanding of the three conditions of self-organization. It takes time for new teams to gain experience in working together as a self-organizing Agile team before they are able to fully utilize the practices that enable self-evaluation and self-improvement—

powering self-transcendence. An Agile team that is able to achieve all the conditions of self-organization including self-transcendence can be said to be at the peak of self-organization.

5.7 Discussion

This section discusses the practices of self-organizing Agile teams with other relevant literature.

5.7.1 Balancing Freedom and Responsibility

Our research suggests Agile teams need to balance between freedom provided by senior management and their own ability to display responsibility in order to achieve and sustain autonomy. Moe et al.'s study of Scrum teams suggests that a lack of a conducive environment provided by management led to reduce the external autonomy in the team [113]. Their study found that high individual autonomy proved to be a barrier to self-organization as members preferred individual goals over team goals. In contrast, our cross-cultural study found that the New Zealand's individualistic culture did not negatively affect collaboration and co-ordination on these teams [15]. Some relatively new teams in both India and New Zealand indicated signs of struggling to make use of the freedom to self-assign and take ownership of tasks. These teams faced such initial problems due to being habituated to working in a traditional software development environments as opposed to an Agile environment. This initial inability to balance freedom and responsibility can be a barrier to self-organization.

XP teams have been seen to balance individual autonomy with team autonomy and corporate responsibility [51] which is similar to the self-organizing Agile team practice of balancing between freedom and responsibility found in our research. Collective decision making is a practice that enables self-organizing Agile teams to balance freedom with responsibility. Self-organizing

Agile teams practice collective estimation and planning of the overall project and the individual iterations. They make collective decisions about the team's norms and principles and collective decide on team goals.

Similarly, teams' ability to take responsibility for tasks (compared to being commanded) was found to be a necessary aspect of a conducive organizational culture in another study [157]. Self-monitoring practices have been shown to influence responsibility and ownership in Agile teams [141, 142, 162]. Daily standups and the use of information radiators have been found to increase social answerability and awareness in Agile teams [162]. Studies describe mature Agile teams are highly collaborative and self-organizing in nature, exhibiting responsibility on both individual and team levels [142]. These studies emphasize the importance of story boards in collaborative activities of mature Agile teams [141]. Our research confirms that status report meetings and information radiators used as self-monitoring practices by Agile teams contribute to balancing freedom and responsibility effectively.

5.7.2 Balancing Cross-Functionality and Specialization

Open workspaces enable the practice of group programming, which in turn promotes close communication and collaboration among team members. Our research strengthens the case for open workspaces as an important part of an Agile team culture [19, 142]

Moe et al. also present the results of exploring the teamwork challenges that arise when introducing a self-organizing Agile team [112]. The results indicate that the main challenges to achieving team effectiveness include problems with highly specialized skills and corresponding division of work. Our research confirms their findings that Agile teams need to effectively balance cross-functionality and specialization in order to sustain self-organization.

5.7.3 Balancing Continuous Learning and Iteration Pressure

The practice of retrospectives has been acknowledged as a way to self-evaluate team performance and secure ideas for constant self-improvement [48]. Our research suggests that holding retrospectives is a crucial practice that enables self-organization in Agile teams. The practice of retrospectives enables the teams to balance continuous learning with iteration pressure, which leads to self-transcendence—one of the conditions of self-organization in Agile teams.

Our research shows that constant learning and improvement in Agile teams is powered by the practices of learning spike and pair-in-need. XP describes a practice called spike solution, which is a simple program to explore possible solutions to complex technical and design problems [19]. A spike solution is used to help estimate challenging and complex user stories and is often discarded after use. A learning spike, as described in section 5.3.2, although not limited to a piece of code, is a similar concept. A learning spike is the extra time taken by the team in an iteration (or spread across a few iterations), specifically to learn new technologies or tools better to perform their tasks.

The practice of pair-in-need provides a collaborative environment that particularly supports newcomers on the team. Newcomers usually feel overwhelmed and lost in a new project [47]. The presence of a mentor has been found to be extremely beneficial for getting newcomers better oriented and settled into their teams [47]. Our research suggests that the presence of a *Mentor*—either an Agile Coach or an experienced team member—and the practice of pair-in-need, when done with a newcomer and a mature team member, help newcomers settle into teams with greater ease.

Pairing has been described as a mechanism for learning through conversations between pairs [131, 164, 165]. Studies have acknowledged that pair programming can be exhausting [131, 51, 165]. Our research found that teams practice Pair-in-Need instead of compulsory, consistent pairing. Teams found Pair-in-Need to be a useful way to achieve learning while managing the pressures of delivering team goals 5.3.

Finally, rotation of team membership has been suggested to help distribute knowledge [119]. The use of rotation to promote knowledge-sharing and consequently cross-functionality is supported by our research 5.2.

Chapter 6

Factors Influencing Self-Organizing Agile Teams

This chapter describes the two critical environmental factors influencing self-organizing Agile teams, that emerged from this research. These factors are: senior management support and level of customer involvement. First, senior management support is discussed in terms of (a) how senior management influences self-organizing Agile teams and (b) how senior management support can be secured for the establishment, functioning, and propagation of these teams. Second, the level of customer involvement is discussed in terms of (a) how different levels of customer involvement influence self-organizing Agile teams and (b) how adequate customer involvement can be secured for the smooth functioning of these teams. Finally, both the factors—senior management support and level of customer involvement—are discussed in light of existing literature.

6.1 Influence of Senior Management Support

Self-organizing Agile teams are greatly influenced by the senior management at their own organizations (P1, P4-P10, P12-P20, P22-P23, P25-27, P29, P31, P33-35, P39-41, P43, P52-53, P55) [83, 78]. The following sections describe the influence of senior management on self-organizing Agile teams, followed by the strategies used by self-organizing Agile teams to secure senior management support at their own organizations.

Figure 6.1 shows the emergence of *senior management support* from underlying concepts. Table 6.1 presents an overview of the influence of senior management and the various business drivers (factors that motivate business decisions) used to secure their support.

Table 6.1: Senior Management Support

Influence of Senior Management
Organizational Culture
Negotiating Contracts
Financial Sponsorship
Resource Management
Securing Senior Management Support via Business Drivers
Applicability to Project Context
Time to Market
Customer Demands
Process Improvement

"..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

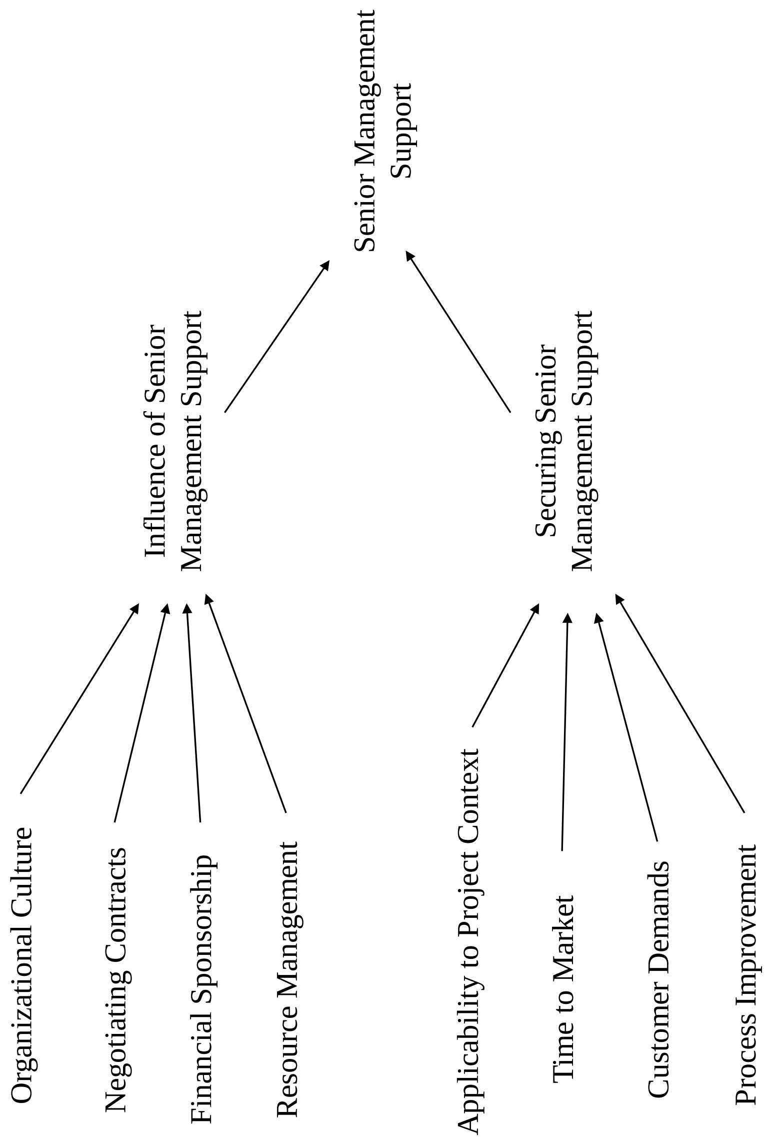


Figure 6.1: Emergence of the category *Senior Management Support* from underlying concepts

> *management has no interest in Agile, they have no interest in what Agile is.*" — P43, Scrum Trainer, India

Senior management influences organizational culture, the types of contracts governing projects, financial sponsorship, and resource management [83]. A senior management that does not support self-organizing Agile teams causes several challenges for the team in each of these areas.

6.1.1 Organizational Culture

Organizational culture has been defined as "*a standard set of basic suppositions invented, discovered or developed by the group when learning to face problems of external adaptation and internal integration*" [135]. Organizational culture has a strong influence on the ability of an Agile team to be self-organizing.

Traditional software development teams typically adopt strictly hierarchical organization structures. Self-organizing Agile teams on the other hand, require organization structures that are informal in practice, where the boundaries of hierarchy do not prohibit free flow of information and feedback. In an informal organizational structure, the senior management is directly accessible by all employees (maintaining an 'open-doors' policy), and accepts feedback—both positive and negative.

Agile organizations, where all the teams operate using Agile software development, are characterized by informal organizational structures. Informality in organizational structure promotes openness. Openness was one of the most common traits mentioned by participants, that made the organizational culture conducive for Agile teams. In such organizations, team members are free to voice opinions, raise concerns, seek management support in resolving their concerns, make collaborative decisions, and adapt to changes in their environment. This freedom provided by senior management is crucial for the team to achieve and sustain autonomy (section 5.1.)

> "*don't expect that you're going to be in any other traditional hierarchical company...no matter if its 4 years or three years [of experience], they [team] can walk up to [CEO's name] and say 'this what you did, is bullshit' (laughs) and [CEO] will say 'oh, OK fine, let's discuss what happened'. So people have that freedom to voice their opinion very clearly. At the same time people will [give] feedback to you.*" — P52, Human Resource Manager, India

Starting with an informal structure has a cascading effect. Informality in the organizational structure leads to openness marked by free-flow of communication and feedback, which in turn leads to an organizational culture of trust. An organizational culture where teams trust their senior management to support them, and when senior management trusts the teams to perform and display responsibility, makes for fertile grounds for self-organization to emerge.

> "*one of the big things that's made a difference there, is they already had an* **environment of trust**. *There was no fear in the organization. You often see a level of fearfulness in very bureaucratic organizations, people are not prepared to give people—to give bad news, you know, the automatic punishment for being the bearer of bad news. I didn't see any of that at [company name], the level of confidence, the level of trust between management and the people on the ground was quite high already. So I think the ground was fertile for Agile...And that was because of the management attitude and the supportive nature of the managers.*" — P26, Agile Coach, New Zealand

In contrast, an organization with a strict hierarchical structure is not conducive for self-organizing Agile teams. A common example is that of a government sector organization, with a strict hierarchical structure. The software development teams in such organizations form one of the lowest levels of hierarchy, topped by middle management, and then senior management. Such hierarchical structure is often coupled with heavy processes, such as heavy documentation, long change management processes, and long software delivery and deployment processes. Such a culture restricts both the team's ability to practice light-weight Agile methods, and their ability to self-organize.

A strict hierarchical structure also has a cascading effect. The hierarchy in such an organization enforces a lack of openness marked by restricted and indirect lines of communication and feedback, which in turn leads to an environment of fear. Teams are afraid of voicing opinions, raising concerns, making collaborative decisions, and adapting to changes in their environment.

> "*...government business drivers are not 'time to market' or producing anything useful...the documentation is definitely more important than actual working software. They are not impressed at all by demos and working software—they almost didn't care! 'Why don't they have a big up front design document?' It basically took me ages to basically force them to accept vertical slicing of that. I think its a fear of giving up control. Control doesn't exist, but they are afraid to give it up ... I was the PM on that project, they are still working on it, I went away screaming!*" — P23, Agile Coach, NZ

On the other hand, some government sector organizations find that their culture, while seemingly different, can be receptive to changes brought on by Agile methods.

> "*It's interesting because it's [Agile] probably a much better fit [to our culture] than you might think. On one hand our organization, part of the culture is that people do tend to work in isolation...But because it's very scientifically oriented there's quite an openness to sharing ideas and information as well...once they [in-house*

> *customers] were exposed to the Agile development group and they were sitting in the room with them and the whiteboard and things, they became very open and very communicative. They would have never have volunteered that or expected that, but once they had people around them that were used to operating that way they were very open to that. So it fit quite well is what I'm saying, it fit pretty well.*" — P18, Senior Management, NZ

Senior management support, in terms of providing freedom and establishing an organizational culture of trust, is therefore extremely important for self-organizing Agile to establish and flourish. A senior management that supports self-organizing Agile teams will (a) maintain an informal structure, (b) provide freedom for teams to provide feedback, and (c) create an organizational culture of trust.

6.1.2 Negotiating Contracts

Self-organizing Agile teams are influenced by the type of contracts that govern their projects [83, 73]. Senior management—either directly in smaller organizations, or through their sales department in larger organizations—is responsible for negotiating contracts with customers. A customer can demand a fixed-bid contract where the cost, time, and scope of the project are fixed up-front. If senior management accepts the customer's demand for a fixed-bid contract, it has far-reaching consequences for the self-organizing Agile team. Teams find that "*fixed price doesn't work well with Agile*" because "*Agile talks about embracing change [and] can't do fixed price projects with changes coming in*" (P42, P27).

The process of fixing the cost, time, and scope of the project in a fixed-bid contract involves estimating the project. A senior management that does not support self-organizing Agile teams, fixes the cost, time, and scope based estimates provided by managers, rather than the teams. As a result the team may be placed under pressure to deliver to often unrealistic estimates. The negative consequences of a fixed-bid contract in an Agile project are captured in the following comment by an Agile trainer and coach who worked several with Indian organizations:

> "*The whole premise of the fixed-bid contract is that requirements will be fixed. The nature of software development is that requirements are inherently unstable and so when you are entering into contract negotiation, you are dealing with the recognition that the requirements will be unstable... Biggest source of dysfunction is not actually from the customer—the greater source of dysfunction comes from within the organization where the contract—fixed bid contract—is negotiated by the sales team, it is negotiated for the smallest amount of money possible. And so the team from day one is under pressure to over-commit and under-deliver and that I see again and again and again!*" — P43, Agile Trainer, India

In contrast, a senior management that is aware of the negative consequences of fixed-bid contracts on the teams better supports self-organizing Agile teams. They provide customers with options. These options include offering an iteration on a trial basis, the flexibility to buy more iterations or terminate the contract with an iteration's notice, and swapping features. For example, an Indian senior manager encouraged customers to buy a few iterations, instead of signing one contract for a large project:

> "*Most of the time...[we] sell a certain number of iterations.*" — P34, Senior Management, India

By allowing the customers to use Agile on a trial basis, Agile practitioners are able to build confidence among customers and provide them with risk coverage. Once the customers have tried a few iterations, then they are offered the option to buy more iterations or features as needed:

> "*One thing we [development firm] used to do and worked very well—we used to tell the customers you don't have any risks...in*

> *case of Agile we enter into a contract with the client—OK we'll show you working software every fifteen days, you'll have the option of ending the project within one sprint's notice. Maximum they can lose is one sprint. Advantage we show to client they don't have to make up their entire mind. . . [they] can include changes in sprints -they see it as a huge benefit to them.*" — P27, Developer, India

Some Agile practitioners allow the customers to swap features. The project is delivered at the same time and price as initially specified in the contract, but the customer can remove product features that they no longer require and replace them with new ones (requiring approximately equivalent effort) that are of more business value to them:

> "*. . . customer after seeing demo after fourth iteration realizes the features built, say the thirteenth feature, is not required and he needs something else. . . he can swap the two.*" — P27, Developer, India

By providing the customers with the option to quit the project in the worst case scenario, some of their financial risks are covered. So if the customers are unhappy with the results, they could always quit the project.

If a customer is still insistent on a fixed-bid contract, the senior management can support a self-organizing Agile team by inviting the team to estimate their projects. Based on the rate of development per iteration—the team velocity—as a guideline, the team can estimate the time required for developing a particular set of requirements in a given domain. Then some amount of extra time could be added to the estimated time as a buffer. The contract is then drawn on this estimated time (including buffer) for a fixed price and scope.

> "*Agile will not ask you in how much time will you [need to] complete the project...but [the customer will]. Sometimes you've got*

> *to map internal Agile practices to customer practices....Actually it comes from a lot of experience on Agile. When you know that okay this is generally the velocity of the team that the team is able to do within the given domain, the given complexity and then you make some rough estimates, including some buffer. [Customer says] 'okay I want these features, tell me the time'. so then we'll make prediction based on Agile data that this is the team size, this is the velocity, we assume the team won't change then the Agile burndown chart will say let's say 2 weeks so we'll say okay another 2 days of buffer, so 2 weeks ands 2 days, something like that.*" — P28, Developer, India

A small amount of buffer time was important to allow the customer the possibility of introducing changes in requirements along the way while giving the development team time to respond to those changes. Buffering was a practical strategy of working with a fixed-bid contract while using Agile methods.

Finally, senior management in Agile organizations are very careful about negotiating contracts that are "*Agile-friendly*". They frequently have a specialized sales team that understand Agile methods and the consequences of the contract on the self-organizing Agile teams.

> "*In the sales room, even the way we work is Agile. We have two groups, one for marketing, one for sales. We have stages for each teams—we use kind of post-its and put them up. So even our sales is Agile.*" — P33, Sales Manager, India

A senior management that supports self-organizing Agile teams will (a) try to convince customers to try flexible contract options, (b) engage the team in providing estimates for the fixed-bid contract, along with adding a contingency buffer, or (c) negotiate "*Agile friendly*" contracts.

6.1.3 Financial Sponsorship

Self-organizing Agile teams need financial sponsorship from their senior management in the form of Agile training and an infrastructure that's conducive to self-organizing practices [83]. The importance of a *Mentor* in the early stages of becoming a self-organizing Agile team has been discussed in chapter 4. The team needs senior management support in order to benefit from the presence of a *Mentor* in the form of an Agile Coach. The Agile Coach is often a contracting consultant, hired specifically to train a new team on Agile principles, values, and practices. In other cases, an existing project manager in the organization may take up the *Mentor* role. The senior management provides financial support by either hiring contracting Agile Coaches or sponsoring these managers, and occasionally other team members, to receive Agile training (e.g. a Scrum Master Certification).

Financial support is also required in the form of infrastructure support, such as setting up an open-plan workplace and tools for electronic communication and collaboration with distant customers. A supportive senior management champions the cause of self-organizing Agile teams and provides financial support for such an infrastructure.

> "*In most organizations I'd say Skype would be blocked. They [senior management in non-Agile organizations] say we do chat or call their friends abroad and waste time but here in [this organization], Skype is there on every machine because the management knows that it is an important communication tool...So yeah definitely the change in the mind-set of the organization has to be there. For example, they [senior management] have provided LCD TVs within the rooms and there are a lot of Skype meeting rooms which have LCD TVs, camera, and you have Skype installed. If I stand up, you actually go through those moves and you can see the customer and they can see us, so like that. Again there is that initiative from the senior management because they might as well*

> *say that 'okay do it on your own machine or we cannot provide LCD TVs for every team!' So that drive has to come from them definitely.*" — P29, Developer, India

> "*...level of sponsorship means...the senior manager...say 'This is the methodology we are adopting. I expect you to change your practices and techniques to support that, and here's some money to do so...here's some time, here's some resources.*" — P7, Agile Coach, NZ

A senior management that supports self-organizing Agile teams is willing to make such financial investments as (a) hiring a *Mentor* for new teams or providing existing Project Managers with Agile training and (b) providing the infrastructure necessary for effective functioning of the self-organizing Agile teams.

6.1.4 Resource Management

An important influence of senior management is the way they manage resources [83]. For self-organizing Agile teams, dedicated resources are highly desired. When team members are allocated to multiple projects, it has a negative influence on the teams' ability to perform and self-organize. One of the main characteristics of self-organizing Agile teams is high levels of cohesion and collaboration within the team. The team's ability to self-organize is dependent on understanding each others' strengths and weaknesses and forming a team culture of openness and respect. It takes time for a team to learn about each other and self-organize based on the members' myriad abilities.

> "*What I think affected our project...[the developer] was working on another project, he didn't have enough time, so he didn't have the space to chat with anybody, to discuss ideas with anybody, to work with anybody, so he was really just on his own, and I*

> *think that really impacted a lot of the work he did in the last few months ... When you're working in a team like this [Agile team] and you've got to work quite closely, the individuals in the team matter.*" — P21, Customer Rep, NZ

If the members are split across multiple projects, it affects their ability to perform group programming that enables self-organization. A senior management that does not realize the implications of their resource management can have a negative influence on the team:

> "*[explaining how resource management works]...resource-assignment, right...If I am VP (vice president)...for me, resource is a pure mathematical figure. 0.25 is 2 hours. if I divide, make the equation work, I'll be happy! Ground reality is different. People can't work 0.25! One side am a VP I want to get business, I have to do equations: 0.5 from here, 0.5 from here etc and make it 3...pure mathematics...not feasible in ground reality...People have to be mature enough...[its] just a matter of understanding the ground reality: if they [senior management] are a developer how would they react to the situation?*" — P39, Agile Coach, India

On the other hand, a supportive senior management values their teams and respects their human side as much, if not more, than their technical skills:

> "*...I personally feel it's one of those companies where does a lot for the people. They [senior management] definitely understand people, values, and you know, they understand their emotions...so we do respect people and you know if they [team] have any concerns or worries we [company] will try to understand it.*" — P52, Human Resource Manager, India

Resource management in terms of the hiring process and removal of individuals from teams is also influenced by senior management. In Agile organizations where senior management supports self-organizing Agile teams, their Human Resources department is set up specifically to hire people that are likely to "*fit*" into Agile teams (section 4.6).

Sometimes, team members need to be removed from an Agile team because of their inability to fit into the culture. One of the team members typically takes on a *Terminator* role and seeks senior management support in removing such individuals (section 4.6).

Senior management supports self-organizing Agile teams through managing resources by (a) providing dedicated resources to projects, (b) hiring individuals to fit into an Agile culture, and (c) removing individuals who threaten self-organizing teams with the help of a *Terminator*.

6.2 Securing Senior Management Support

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

In non-Agile organizations, a pilot Agile team must secure senior management support in order to survive. One of the team members typically takes on the role of *Champion* to secure senior management support. It is important that the *Champion* understands their *business drivers*—the factors that motivate senior management's business decisions. Using these drivers, the *Champion* convinces senior management at their organization to support self-organizing Agile teams. Some of the business drivers or motivators include applicability of method to project context, time to market, cost-effectiveness, customer demands, and process improvement.

6.2.1 Applicability to Project Context

From a senior management perspective, Agile methods are one of several methods from a tool-set 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, [but] where there's batch systems processing data and spitting out there's relatively less opportunity for interaction to demonstrate the outputs...so it'll mean paying a bit more attention to how they get feedback and how the iterations occur, where does the confirmation come from...I think particularly anything that has any kind of user interface for example, which is more than trivial, Agile is a better way to go ... 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 [75]. A *Champion* is cautious not to advocate Agile irrespective of project context—an effort that can eventually backfire.

> "*...recognising that Agile does not deliver to every type of project. So for example, I'd have trouble understanding how you could*

> *do an iterative development of an infrastructure project. In the sense that, you know for a web development, you could do the login screen before you have the database, to capture user data, see what it looks like, say 'yeah I'm happy with that', and move on. 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.

6.2.2 Time to Market

Time-to-market is another important driver for a senior management. In the present world of fast-paced development, cut-throat competition, changing customer requirements, and businesses thriving on innovation, the time it takes to develop and deploy a product to the market is an important driver for senior management. Faster time-to-market is one of the advantages of Agile methods showcased by a *Champion* in a bid to convince senior management. A *Champion* explains how self-organizing Agile teams are able to produce working software iteratively and incremently such that changing customer requirements and latest business trends can be accommodated easily. They also highlight how Agile allows them to eliminate waste by focusing on customer priorities, which in turn leads to a shorter time to market.

> "*You talk to the business in terms that matter to them...Getting them to realise that 60% of specified software is useless, no one actually uses it. [Something] might seem like a good idea but when you look over people's shoulder at what functions they're actually using, most often its only 20% of what is specified as*

> *frequently used, and maybe another 20% that they sometimes use. Getting them to realise that and asking them what's actually really important from them to get from A to B. What's the minimum you can get away with.*" — P20, Agile Coach, NZ

Traditional hierarchical organizations often have heavy documentation processes. The time spent in lengthy up front documentation can be a huge waste in the face of changing requirements and can easily slow a product's time-to-market. When senior management realizes this problem, they are more willing to invest in Agile methods that offer just enough documentation and faster time-to-market [77].

> "*They were a very successful organization, they built award winning products...they won the [name] innovation award for...the best [name] product for 2009. Building that product nearly killed them...the team was exhausted. [Senior executive's name]...[had a] look at the real numbers, they had 4 man years worth of effort into building a requirements document...They looked back at this requirements documentation and they looked at the product. What was actually in the product, versus the requirements document: 25% of the requirements that were identified in the document were in the delivered product, and they accounted for only half of the functionality of the product. So, 75% of the work they had done was wasted, because it had all changed!...[Senior executive's name] sent herself and 6 other people came along to hear about this Agile stuff...After that, they went away and did a whole lot of thinking, and decided, yeah let's try it.*" — P26, Agile Coach, NZ

Fast time-to-market is one of the best cards a *Champion* can play, because faster delivery of working software is one of Agile's most commonly claimed advantages.

6.2.3 Customer Demands

While some customers are skeptical about Agile methods, other may specifically demand an Agile approach to developing their projects. With the growing popularity of Agile methods in software industries around the world, more customers are looking to engage in Agile projects. This is specially true in the context of the Indian Agile teams catering to customers in North America and Europe.

> "*You know, in part I think because customers don't really understand that it means but it is a huge buzzword right now. Its the big thing. There is nothing bigger in the software world right now really.*" — P43, Scrum Trainer, India

Responding to their customers' demands is an important driver for senior management. As a result, they encourage their teams to learn Agile methods in response to customer demands. A related problem is that some senior management mandate an Agile approach in response to customer demands, but do not understand their own role in the process.

> "*Sometimes there's a mandate from top that we all go Agile but the problem with that approach is that they give a mandate but they don't give an environment for a self-organizing team to start working.*" — P31, Agile Coach, India

A senior management that understands their own role in the process, not only mandates Agile projects in response to customer demands, but also changes their own practices to support self-organizing Agile teams.

6.2.4 Process Improvement

A choice of software development method can be driven by a need for process improvement. Senior management in organizations with no well-defined software development process are often easier to convince to try Agile (P17, P36). In contrast, senior management in organizations used to traditional software development methods, need to be shown a marked improvement brought on by the introduction of Agile methods. To convince senior management, the *Champion* collects and reports metrics to demonstrate process improvement. The metrics available in regular Agile projects are very different from those in non-Agile projects, so senior management can harbour misconceptions about the new metrics and struggle to understand them.

> "*...[Agile] provides a set of reports that current senior management...do not understand. What the hell's a burn up chart? what is a burn down chart? What's velocity mean? What do you mean by story cards?...[some] people, unless it's in a Gantt chart, cannot see it as a project...[senior management asks] 'you're writing on bits of paper to plan a project?' So one of the things we did with [organization's name] was we had printed ones you know, some companies do their logos and that helps, gives it that more air of self-importance...It's just that [some] people look at and say, 'oh its got a printed card, it must be a proper process'.* " — P7, Agile Coach, NZ

Senior management may relate formality with robustness, and so several Agile processes and artifacts that are paper-based and informal may appear less robust. An effective *Champion* understands the importance of translating Agile metrics into traditional metrics in the early stages of transitioning, so that their senior management can comprehend them and evaluate the performance of the pilot team.

> "*You know, early on, you might want to do some sort of translation. Whether that be a series of two lines on your Gantt chart, which gradually drop down as the project goes over time with the set of features...And slowly say, 'and actually this means this in this part of the graph' and wean people off the old methodologies into the new.*" — P7, Agile Coach, NZ

Initial translation between Agile and traditional metrics allows senior management to ease into the process. While it takes time and effort to read and understand reports with two different metrics, it is a valuable long-term investment.

> "*What I did receive was two types of report: one's just a financial report saying these are the iterations we're expecting to run, this is our run rate, and the other report was against what we call the loosely termed 'complexity points'...the burn rate of the dollars and the burn rate of complexity points would be equivalent, and so I got reports showing whether or not that would be true.*" — P18, Senior Management, NZ

Senior Management is quick to spot processes that show marked improvement in team performance and effectiveness.

> "*The head of the [name] systems division, [name]...stood up at the end of that day [of estimation and planning] and he said, we have achieved in 6 to 8 hours, what normally takes us 6 weeks. He was absolutely blown away, stunned. And from that point onwards that team was now dedicated and focused to working on this product. They made a huge change in the way that they organised the offices, and they did in fact move people around, so they stopped being developers, analysts, designers, testers, in separate office spaces.*" — P26, Agile Coach, NZ

In the absence of any real drivers for change, 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

> "*I think it's one of those ones where there was no clear mandate for change, there was no reason—they [senior management] didn't see a reason for them to change—and also, that the project was starting to highlight their inefficiencies, which made them uncomfortable ... it's one of the strongest teams in the company. It is difficult and it's quite hard for me.*" — P7, Agile Trainer, New Zealand

Senior management is typically willing to support self-organizing Agile teams through (a) changing their organizational culture, (b) negotiating Agile-friendly contracts, (c) providing financial sponsorship, and (d) managing human resources in a way that supports self-organization, only when they find a need for it. The need to change all these organization-wide processes—which can be expensive, time-consuming, and challenging—is defined by various business drivers, such as (a) applicability to project context, (b) time-to-market, (c) customer demands in response to industry trends, and (d) process improvement. The *Champion* tries to convince their senior management to support self-organizing Agile teams by showcasing the advantage of Agile methods in light of the business drivers. Once senior management is convinced that use of Agile methods rewards their business drivers, they are more likely to make the organization-wide changes required to support self-organizing Agile teams.

6.3 Influence of Customer Involvement

Self-organizing Agile teams are influenced by the level of customer involvement they receive on their projects [73, 74, 82]. Inadequate customer involvement has negative consequences for the team, while adequate customer involvement has positive consequences for the team. This research found several influences of customer involvement on teams and multiple strategies for securing customer involvement [74, 82]. One way the customers influence self-organizing Agile teams is through negotiating fixed-bid contracts. Customers demanding fixed-bid contracts place limitations on the team's ability to respond to changes. The main influence of negotiating contracts is a pressure to over commit. The influence of negotiating fixed-bid contracts and the strategies of providing flexible contract options and buffering have been discussed at length in section 6.1.2.

The following sections describe the most critical influences of customer involvement on self-organizing Agile teams, followed by the some of the most popular and innovative strategies used by the teams to secure customer involvement. Figure 6.2 shows the emergence of *level of customer involvement* from the underlying concepts. Table 6.2 presents an overview of the influence of customer involvement and the various strategies used by self-organizing Agile teams to secure adequate involvement discussed in detail.

6.3.1 Gathering and Clarifying Requirements

Customer representatives are meant to provide requirements in the form of user stories every iteration [138]. They are also responsible for clarifying these stories for the development team as needed. In real-life Agile projects, however, development teams faced challenges in retrieving requirements from

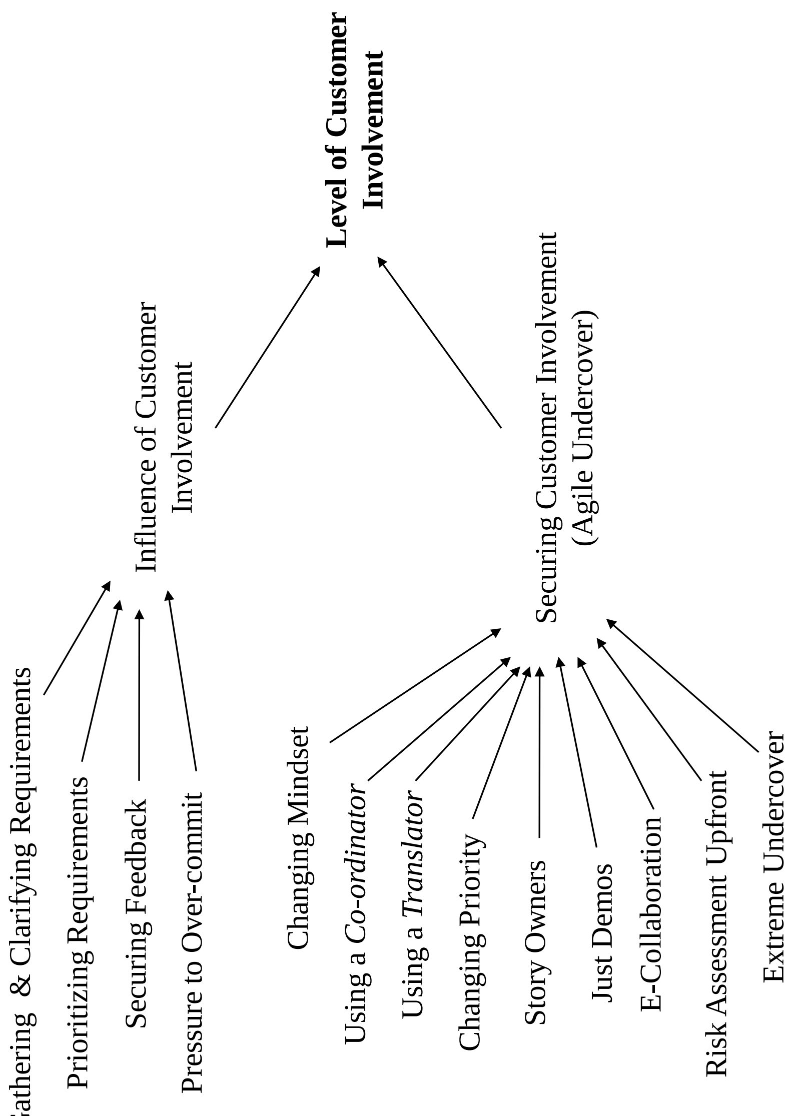


Figure 6.2: Emergence of the category *Level of Customer Involvement* from underlying concepts

Table 6.2: Level of Customer Involvement

Influence of Customer Involvement
Gathering and Clarifying Requirements
Prioritizing Requirements
Securing Feedback
Securing Customer Involvement
Changing Mind-set
Changing Priority
Story Owners
Just Demos
e-Collaboration

customers:

> "*To get requirements from the [customers]...was one of the worst things in this project, honestly! We'd be sitting there for two weeks waiting for an answer.*" —- P4, Business Analyst, New Zealand

> "*The biggest frustration I had on this project was that...we don't have the [customer representatives] that we can gather requirements from.*" — P1, Developer, New Zealand

Inability to gather requirements in time for iterations could result in unnecessary delays and loss of productivity:

> "*We are extracting our requirements just in time from the business - the detailed requirements. It would be impossible if there was no full time person inside the project it would get stalled.*" — P10, Agile Coach, New Zealand

> "*The team has the capacity...[but] with Agile if you don't have the requirement you can't do anything...because you are supposed to be in-line with business.*" — P1, Developer, New Zealand

Similarly, some teams have issues trying to get customer representatives to clarify requirements:

> "*Things [awaiting clarification] would queue up for them and then they'd just answer the whole queue at once...then as soon as they got busy again it would start to get a bit harder.*" — P11, Developer, New Zealand

Without clear requirements, teams are forced to make assumptions about the customer's needs and priorities:

> "*In the absence of business requirements from customers, the teams make assumptions and get misaligned from the desired business drivers. The result is a product or feature that is not aligned to the perceived business requirements.*" — P10, Agile Coach, New Zealand

These inaccurate assumptions lead to the team building features that are not as per the customer's intended requirements. The teams would then have to perform rework which incurs additional costs for the customers.

> "*So from my perspective as a developer, yes, the more the client is involved, the better for us...But I've seen projects in the past where we had to redo all the components and it was very expensive basically to the client because we were being paid [for rework]*" — P14, Developer, NZ

Rework is both costly to customers and taxing for developers if it has to be done at a later time. Due to delays in customer feedback, the need for rework typically does not surface until much later, by which time it is difficult for the developers to return to a particular story and rework it.

> "*Yes [we had to rework] but it's not the re-work, it's re-worked easily as long as it's near the time you did it. So having to go back and augment what you did three weeks ago was [hard].*" — P11, Developer, New Zealand

As a result of insufficient and ineffective customer involvement, the development teams were unable to get customer representatives to provide and clarify requirements.

6.3.2 Prioritizing Requirements

Agile methods require customer representatives to prioritize the order in which the team should work on the user stories, driven by business value. Understanding and using the concept of prioritization doesn't always come naturally to customers new to Agile projects:

> "*[customers have to be involved...the customer needs to tell his priorities that this is the first thing we want.*" — P28, Developer, India

> "*We'd just get a whole lot of requests sent at us, by phone and email and all different ways and it took a long, long, long time for them to understand that we needed them prioritised so we knew what was the most important to be doing.*" — P6, Agile Coach, New Zealand

> "*We're meant to have one list of product backlog and it's supposed to be prioritized but when the client says 'Oh that's all priority' we have to go back and say 'which?! what do you mean?!...you can't have all priority!'*" — P2, Developer, New Zealand

Some teams face difficulties in getting customer representatives to prioritize the requirements and as such the teams are confused about what features to develop and deliver first. In contrast, teams that receive adequate levels of customer involvement are able to gather and clarify requirements from their customers more easily and effectively.

6.3.3 Securing Feedback

Customer feedback is of vital importance in ensuring the desired product is being developed and delivered incrementally. As a senior developer pointed out "*the whole point of the two week iterations was so that the end users could know if we were on the right track*" (P25) and requires the customer representatives to provide feedback on developed features.

> "*If [the customer representative] didn't respond you just didn't care about their opinion...and at the end of the project...the business units 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, NZ

In absence of customer feedback, some teams are unable to assess how well the features meet the requirements. In contrast, teams that receive adequate levels of customer involvement have better, more direct, and more frequent communication with their customers. Examples of direct and frequent collaboration with customers has been discussed in section 4.2.

Finally, the importance of adequate customer involvement is summarized by a customer representative themselves, in the following comment:

> "*Well I'm sorry, if you're not prepared to take one person out of their job for two weeks and put them in an office doing nothing but answering questions about a [product] they're building for you, you deserve what you get!*" — P9, Customer Representative, NZ

6.4 Securing Customer Involvement

Several interesting strategies are used by the teams to secure customer involvement. These strategies are collectively named *Agile Undercover*, a category that emerged from the data analysis [74]. These strategies include: Changing Customers' Mind-sets, Providing Options, Buffering, Changing

Priority, Risk Assessment Up Front, Story Owners, Using a *Co-ordinator*, Using a *Translator*, Just Demos, E-collaboration, and Extreme Undercover. Agile Undercover strategies allow teams to successfully secure customer involvement in some cases and continue to practice Agile in the face of inadequate customer involvement in others.

Risk Assessment Up Front is a general strategy for assessing the risks involved in an Agile project up front. The level of customer involvement was one of the risk items assessed using a risk assessment questionnaire. Extreme Undercover was a strategy used by some teams to practice Agile internally while appearing to be a traditional software development team to the customers. This strategy was found to be used by some Indian teams in the initial stages of adoption, where they faced extremely skeptical customers. Over the course of the research, as the popularity of Agile methods increased, this strategy was rarely observed. Providing options and buffering have already been described in section 6.1.2 and also presented in [74, 82, 75]. A *Translator* was used to overcome the language barrier between teams and their customers. A *Co-ordinator* was used to help co-ordinate customer collaboration and change requests across distances. The *Translator* and *Co-ordinator* roles have been discussed at length in sections 4.3 and 4.2. A description of these strategies is available in our publications on this topic [78, 73, 74, 82], they are not reiterated here for space reasons. The following sections describe the rest of these Agile Undercover strategies used by Agile teams to secure customer involvement.

6.4.1 Changing Mindset

A *Promoter*'s role in convincing customers to try Agile methods and collaborate with teams is extremely important. Some customers harbour skepticism about Agile methods and are unwilling to extend collaboration. A *Promoter* tries to change the mindset of such customers by explaining the principles and values of Agile methods.

> "*The people [customers] who are coming from typical bigger companies they would have read about it or have the wrong idea of Agile. We interactions with them, have a series of talks...and explain to them what Agile is.*" — P36, Agile Coach, India

One of the participants, who played a *Promoter* role, highlighted the advantages of Agile methods to their customers in a bid to secure their involvement.

> "*...focus is on delivering business value as soon as possible - as a result of that you take items which are most required from point of view of business, not the ones that are most interesting in terms of technical implementation.*" — P27, Developer, India

A *Promoter* asserts that frequent customer involvement allows customers better control of the product. A constant focus on customer priorities was seen as an advantage by customers, many of whom became willing to get involved in the process. As one of the customer representatives, convinced about the advantages of Agile methods, revealed:

> "*...when it's done correctly, [Agile] makes Waterfall look archaic. As a business owner or a business representative, the control that you have and the ability to change your mind and to keep the project abreast of things that are going on in the business, is unparallelled.*" — P9, Customer Representative, New Zealand

Finally, in a bid to change customers' mindsets, some Agile organizations offer Product Owner training to their customers in order to familiarize them with their responsibilities as an Agile customer.

6.4.2 Changing Priority

In an effort to maintain the iterative and incremental nature of their Agile projects, teams are forced to lower the priority of user stories that are awaiting customer requirements, clarification, or prioritization. Such stories are usually demoted in priority and pushed further down into the product backlog until the required customer response is secured and development on those stories can re-commence. Agile teams confess that they change the priority of the story in absence of enough, clear, and prompt requirements (P1, P8, P14, P22, P30).

> "*[If] we know exactly what business want or we know 80 % of what they want, we include that story in the sprint; otherwise if we have something that's a little bit unsure, we don't include that in the sprint.*" — P1, Developer, NZ

A similar strategy, called *definition of ready* was adopted by an Indian team [18]:

> "*We have recently started using...the definition of ready.....product owner will not take something that is not 'done' and similarly developers are not going to take something that's not 'ready'.*" — P30, Developer, India

A user story was considered *ready* when the customers had provided the business goals and expected outcome associated with the story and implementation details necessary to estimate the story had been discussed. A story that was not *ready* was not able to achieve priority in the product backlog.

6.4.3 Story Owners

In absence of the on-site customer, Agile teams use *Story Owners* where members of the customer organization share the responsibility of the customer role and are available as and when required. The practice of assigning Story Owners was an adaptation to the Scrum practice of allocating a product owner [138]. Story owners are responsible for particular stories (less than a week long), instead of *all* the stories in the product backlog: "*every story* *had to have an owner to get into prioritisation.*" (P14) Assigning story owners serves a three-fold purpose. Firstly, having multiple story owners instead of a single customer representative for entire project means no one person from the customer's organization is expected to be continuously available. This lessens the burden of the customer representatives, who have their own operational jobs to tend to alongside playing an Agile customer.

> "*We didn't need that story owner for the duration of the project, we normally only need them for part of an iteration.*" – P22, Agile Coach, New Zealand

Secondly, it allows the team to plan out stories for development in synchronization with the corresponding story-owner's availability. Thirdly, it encourages a sense of ownership among customer representatives as they are encouraged to present their own stories to peers at end of iteration reviews.

> "*We get the [story owners] to demonstrate those stories to their peers at the end of the iteration review, this concept is something we've evolved over the project.*" — P22, Senior Agile Coach, New Zealand

After one such presentation a particularly skeptical customer representative was "*quite chuffed [pleased], and at the [next] iteration planning meeting, that person was all go! Instead of sitting back with their arms folded, they had their elbows on the table, leaning forward, and were driving the story detailing conversations we were having.*" (P22)

6.4.4 Just Demos

Demonstrations are used by Agile teams as a powerful mechanism to secure the much needed and elusive customer feedback. The team presents working software to the customer representatives at these regular demonstration meetings and receive feedback from them regarding the features delivered in that iteration. This feedback is then incorporated into the development cycles.

> "*Often there's someone from each of the [customers] have a look to see what were doing and how it will affect them.*" — P6, Agile Coach, New Zealand

Using demos provides the opportunity to clear any assumptions made by development teams as a consequence of the customer representative not providing enough or clear requirements:

> "*you are communicating more generally with the client by virtue of the fact that if nothing else you are releasing software more frequently in iterations to the client....Developers have their interpretations of what is that they are supposed to be doing. What we try to do to mitigate that is frequent working software that we get in front of the client and we say this is what we think you want and they say that's not even close! And we say okay cool, at least we know that now rather than at the end of the project.*" — P19, Senior Management, New Zealand

Demonstrations were often the only regular involvement that some Agile teams receive from their customer representatives. The teams use this opportunity to receive feedback and clarifications. The customers appreciate the demos despite their potential reservations about Agile in general because it provides them with increments of working software:

> "*We gave demo after fifteen days. [The customers] liked what we were doing because they were not used to some additional features very fifteen days. We were getting 4-5 people from client organization in the demo. They were pretty impressed with that concept...happy with the results.*" - P27, Developer, India

Teams utilize demos to discuss requirements and get clarifications in addition to receiving feedback on demonstrated features. As the local and involved customer representative of a NZ team disclosed:

> "*Just the sprint demos…and [we see] three pieces of functionality and it's all done in fifteen minutes, we take the full hour to discuss the other things…the demos were fun. I don't know if that's their intent, but they were!*" — P9, Customer Rep, New Zealand

A demo also proved to be a useful way to get collaboration from distant customers:

> "*[distant customers] can't be here every day or every week so we only got to do emailing and phone calls during the demo.*" — P2, Developer, New Zealand

This strategy was found to be useful in securing customer feedback from distant and skeptical customers. Almost all customers are interested enough to attend demonstrations as it gives them an opportunity to see new functionalities of their software.

6.4.5 E-collaboration

Electronic collaboration (e-collaboration) is a popular means of communicating with customers using phone, email, chat, and voice/video conferencing. For Indian teams with off-shored customers, e-collaboration is a practical work-around:

> "*Video conferencing becomes very important. Its all about collaboration [when] time difference is a problem…with Europe [there is a] 4 hours overlap.*" — P27, Developer, India

Some New Zealand teams with distant customers were also seen using phone conferencing with shared documents and emails:

"*[Using] webX...its an online forum and as a host we get to call up documents and share them and they can come in and view.*" — P8, Tester, New Zealand

"*Web-conferencing...chats...[enable] stand-up meetings over the web. You can do demos that way.* — P20, Agile Coach, New Zealand

"*Skype or video-conferencing...doesn't cost that much — to use Skype its literally zero.*" — P1, Developer, New Zealand

With increasing number of software projects being off-shored globally or spread across multiple sites, face-to-face collaboration has become a practical challenge. E-collaboration is a popular alternative used by software teams to overcome this issue because (a) Agile requires regular customer involvement (b) several teams have physically distant customers making face-to-face collaboration difficult and (c) e-collaboration provides a cheaper alternative.

In summary, adequate levels of customer involvement is extremely important for self-organizing Agile teams. Inadequate customer involvement leads self-organizing Agile teams to adopt coping strategies, many of which are not ideal. Teams that receive adequate customer involvement, on the other hand, are able to concentrate on delivering quality products to meet their customers' demands.

6.5 Discussion

Self-organizing teams do not emerge and flourish in isolation [74, 82, 83]. Teams depend on environmental factors such as the support of senior management at their own organization and the level of customer involvement on their projects. Moe et al. identify lack of support system as a barrier to self-organization [113]. Beck notes that an Agile team is not equipped to handle the "*foreign relations*" with the rest of the organization by themselves

[11]. The *Champion* and *Promoter* roles, mostly played by Agile coaches, handled these relationships. The following sections discuss these two most critical factors influencing self-organizing Agile teams, in light of the existing literature on the subject.

6.5.1 Senior Management Support

Senior management influences the organizational structure and culture in an organization [119]. The importance of senior management support in the form of a conducive organizational culture has been widely acknowledged [67, 19, 35, 44, 51, 119, 148, 157]. Agile methods challenge conventional management ideas, and require changes in organization structure, culture, and management practices in traditional software development organizations [51, 119]. Changing mindsets and cultures, however, is no trivial task [29].

Beck highlights the influence of organizational culture on the use of Agile methods and argues that an environment of isolation, timidity, and secrecy will cause challenges [20]. Our research supports the claim that an environment of openness, communication, and trust is imperative for self-organizing Agile teams to function. The influence of senior management in creating and maintaining such an environment is extremely important.

A study of the influence of organizational culture on Agile methods use found correlations between certain aspects of organizational culture and the use of Agile practices [148]. In particular, the study found that organizations that value collaboration, feedback, learning, and empowerment of people are better suited to support Agile methods. Our findings support these claims, as well as the conclusion that hierarchically structured organizations are not well suited to Agile methods. Management in Agile teams is meant to be more facilitative and collaborative [119, 148]. Empowerment and collective decision making in Agile teams are seen to increase their ability to self-organize [119]. Similarly, our research shows that these aspects of organizational culture have a strong influence on the self-organizing ability of Agile teams.

Tolfo and Wazlawick studied the influence of organizational culture on the adoption of XP [157]. Their study concludes that while XP generally assumes the existence of a conducive environment for XP teams, such an organizational culture is not always present in software organizations. In particular, the level of autonomy an organization provides to its members was found to be an important ingredient of a conducive organizational culture. Our findings supports this claim and link senior management support to self-organizing teams.

Most studies that have explored the influence of senior management support and organizational culture have focused on XP teams [130, 157]. Studies exploring the influence of organizational culture on Scrum teams, however, are limited. In a Scrum-based study, Moe et al. found that the management did not provide an environment conducive to self-organization that led to reduced external autonomy [113]. Our research found that self-organizing Agile teams (practising Scrum or combinations of Scrum and XP) require a conducive organizational culture marked by freedom, openness, trust, and an informal organizational structure. In contrast, an organization with a hierarchical organizational structure and an environment of restricted, formal, and indirect communication restricts the teams' ability to self-organize.

In a paper on introducing lean principles with Agile practices in a Fortune 500 company, Parnell-Klabo described various difficulties in securing buy-in for a pilot project [125]. Some of these included obtaining facility space for collocation, gaining executive support, and influencing the change curve. Our research describes how our participants went about securing senior management support (section 6.2).

Several attributes of Agile methods are well aligned with senior management's business drivers discussed in this chapter. For example, fast delivery and rapid response to changes in business and technology is a key attribute of Agile methods [6, 19, 26, 72]. It would appear then that convincing senior management to support self-organizing Agile teams would be an easy task. However, this is not always the case. Organizations don't change for the sake of change, they change when they see benefit from it.

A single case-study of adopting XP at a diverse, multidisciplinary web-development environment at IBM highlights the existence of skepticism amongst senior management regarding Agile nomenclature. For example, the use of the XP term "*planning game*" was not well received by senior executives who preferred more formal-sounding terms like "*planning process*". Section 6.2 provides examples of skepticism faced when trying to secure senior management support. Our findings suggest that convincing senior management not only requires that a team member takes on the role of a *Champion*, but also that they understand senior management's business drivers. In other words, senior management does not undertake drastic changes in their organizations without a strong incentive. Understanding the business drivers particular to different organizations and their senior management is critical for a *Champion* advocating the introduction and continued support for self-organizing Agile teams.

Most of the above mentioned studies have explored the influence of management support on the adoption and use of Agile methods. Our findings show the influence of senior management support on self-organizing Agile teams and highlight various strategies used by teams to secure such support in an effort to achieve and sustain self-organization.

6.5.2 Level of Customer Involvement

Customer collaboration in traditional software development projects is typically limited to providing the requirements in the beginning and feedback towards the end, with limited regular interactions between the customer and the development team [44, 66, 68, 87, 119]. In contrast, customer collaboration is a vital feature [73, 72, 107] and an important success factor in Agile software development [35, 100, 110]. 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 [55, 66, 107, 119].

There is empirical evidence to show that effective customer communication and feedback are critical in Agile software development [90].

An ideal customer representative is an individual who has both thorough understanding of and ability to express the project requirements and the authority to take strategic decisions [44, 55, 66, 119]. Boehm advocates dedicated and co-located CRACK (Collaborative, Responsible, Authorized, Committed, Knowledgeable) customers for Agile projects [29]. Training in Scrum process, has also been advocated for customers in order to better understand their role [104].

Several studies have, however, described a gap between the ideal Agile customer role and the level of customer involvement on Agile projects in practice [31, 42, 44, 90, 103, 127, 131]. These studies have identified varying levels of customer involvement in their own case studies, both in terms of the quality and quantity of that involvement.

Martin et al. found that the on-site customer role in XP projects, although perceived as rewarding by some customers, was largely seen as overburdening and inherently un-sustainable [107]. They discovered that the customer role was played by a team of people, instead of by a single person as initially assumed in literature. Martin et al. describe an informal XP customer team that consist of different roles. Of these different roles, the Negotiator is a customer representative who has in-depth domain knowledge, provides requirements to the development team, and is willing to carry responsibility of project success or failure. The Negotiator role is the closest to the classic customer representative role and interacts directly with the development team. In addition to these qualities, our participants suggested that customer representatives should understand both Agile practices and their own responsibilities in the process of Agile software development (P5, P12, P29). Martin et al. describe certain customer practices such as Customer Boot Camp and Pair Customering. These practices—when combined with the customer roles they identified—can help reduce the burden placed on the on-site customer role and the XP team.

Some customers are unwilling to set aside the amount of collaboration time required on Agile projects, while in some other cases, the customer representative appointed are lacking in knowledge and authority [44].

Conboy et al. analyzed two completed projects through the use of focus groups [42], and noted that the two teams differed dramatically in their assessment of the value of the customer's input in their project. One team consistently rated the on-site customer role as an excellent addition to their set of practices, while the second team consistently rated this role very poor — essentially counter-productive — influence on the project's successful completion. The team that rated the on-site customer role badly did so because the customer was expensive, did not actively participate in many of the key activities, and was only available for at most two hours of the typical working day due to being on a different shift. This reinforces the need for mitigating strategies where continuous and active customer collaboration cannot be achieved.

A customer proxy is often used in situations were customer involvement is not ideal [87, 101, 104]. Grisham et al. report on the use of proxy to supplement a part-time or unavailable customer [66]. Sometimes a proxy may work to support a Product Owner [87]. In this case, the proxy was a member of the team and an experienced Scrum Master. The use of a proxy allowed the Product Owner to fulfil their role with the minimum of time commitment and allowed the team to benefit from the continuous presence and involvement of the Product Owner proxy. A multi-site case study reported project managers acting as customer liaisons [31]. These roles were also referred to as surrogate customers, and occurred during the adoption of Agile practices. In our research, the role of the *Co-ordinator* is similar to the surrogate customer role and acted as the team representative to the customer. Similarly, Mangalaraj et al. explored two projects and identified that one project had no dedicated customer or proxy customer [103].

Another situation where the Product Owner role may be derived from the development team is when the 'customer' is in fact the end-user. Lowery et al. report on experiences in scaling Scrum at the BBC [101]. The 'customer' in this case was the end-user of the internet services provided by BBC's online iPlayer project. As such the role of the Product Owner was delegated to a member from within the different development teams. This was akin to the *Co-ordinator* role in our participants' teams. Our participants agreed that playing a *Co-ordinator* was demanding yet useful in co-ordinating with distant customers (P2, P4, P13-P14, P25, P34-35, P54).

Pikkarainen et al. [127] studied the impact of Agile practices on communication in software development and found that requirements provided by external customers were not always understandable for the developers. Korkala et al. [90] conclude that misunderstood requirements were a reason for late and unreliable software. The *Translator* role identified in our research, specifically helped mitigate this problem of a language barrier between customers and development teams. Using the definition of ready for user stories forced customers to provide detailed requirements with clear business drivers [18]. The definition of ready complemented the existing Scrum definition of done [138].

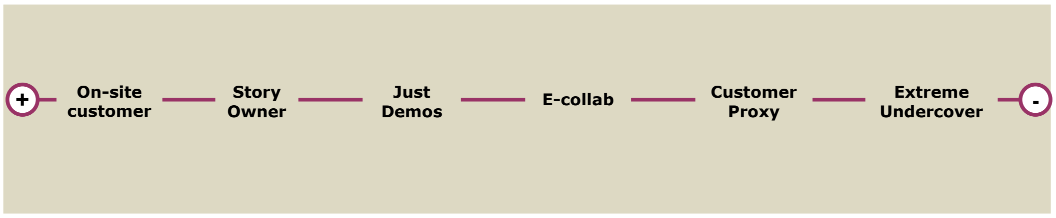


Figure 6.3: Continuum of Customer Involvement on Agile Projects [82]

Our research suggests that there is a continuum of levels of customer involvement on real-life Agile projects. Figure 6.3 depicts the continuum of levels of customer involvement based on the directness of the collaboration. The levels in between the two extremes are not strictly linear and may occur simultaneously, such as *Just Demos* may take place using *E-collaboration*. The continuum assumes that the amount and quality of involvement are the same for all levels. The ideal level is a most direct customer involvement via the on-site customer where the real customer representative is present face-to-face and in person for most collaboration-intensive practices as per Agile guidelines. The practice of assigning *Story Owners* was an adaptation of the existing product owner practice. Unlike the product owner, the story owner was only responsible for one story at a time [100, 138]. This was an effective way of overcoming the limited availability of customer representatives. Story owners also provide an alternative to the practice of on-site customer which has been found to be effective but burdening and un-sustainable for long-term use [66, 90, 100, 107].

This is followed by *Just Demos* where the level of customer involvement is limited to participating in end of iteration demonstrations. Although demos are a regular Agile feature, they were often the only face-to-face collaboration time our participants received from their customers and they used *Just Demos* to discuss features and receive clarifications in addition to feedback.

The next level is *E-collaboration* where the team interacts with the customer representative over electronic means such as video conferencing. Face-to-face communication is considered "*the most efficient and effective method of conveying information to and within a development*" [87, 72], followed by video-conferencing, telephone, and email [90]. Our participants used *E-collaboration* extensively but noted that "*it does not take the place of having somebody sitting beside you*" (P8). Other limitations were imposed by the tool itself, such as Skype not supporting three or more people through video chatting (P1).

This is followed by a *Customer Proxy* from the team playing the role of the customer representative in absence of the real customer involvement; followed by the least desirable level, *Extreme Undercover* where the customer is unaware of the Agile nature of the project.

Chapter 7

Conclusion

This chapter summarizes the main contribution of this thesis—a grounded theory of self-organizing Agile teams. This is followed by a discussion of the relationships between the roles, practices, and factors. The next two sections critique our grounded theory and identify the limitations of this study. This is followed by a discussion of the theory in the light of existing literature, implications for practice, and suggestions for future work.

7.1 Research Contributions

This thesis presents a grounded theory of self-organizing Agile teams. This theory is based on a Grounded Theory research study involving 58 Agile practitioners from 23 different software organizations in New Zealand and India over a period of 4 years. The 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*, in order to become a self-organizing Agile team.

Figure 7.1 presents a diagram depicting the theory of self-organizing Agile teams. The main contributions of this thesis are as follows:

7.1.1 Self-Organizing Agile Team Roles

The self-organizing Agile team roles are:

- *Mentor* that 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.
- *Co-ordinator* who acts as a representative of the team to co-ordinate customer collaboration with the team and manage customer expectations.
- *Translator* that understands and translates between the business language used by customers and the technical terminology used by the team, in an effort to improve communication between the two.
- *Champion* that champions the Agile cause with the senior management within their organization in order to gain support for the self-organizing Agile team.
- *Promoter* that promotes Agile with customers in an attempt to secure their involvement and collaboration to support the efficient functioning of the self-organizing Agile team.
- *Terminator* that 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.

The informal, implicit, transient, and spontaneous nature of these roles are characteristic of self-organizing teams [10]. Detailed descriptions of these roles are provided in chapter 4.

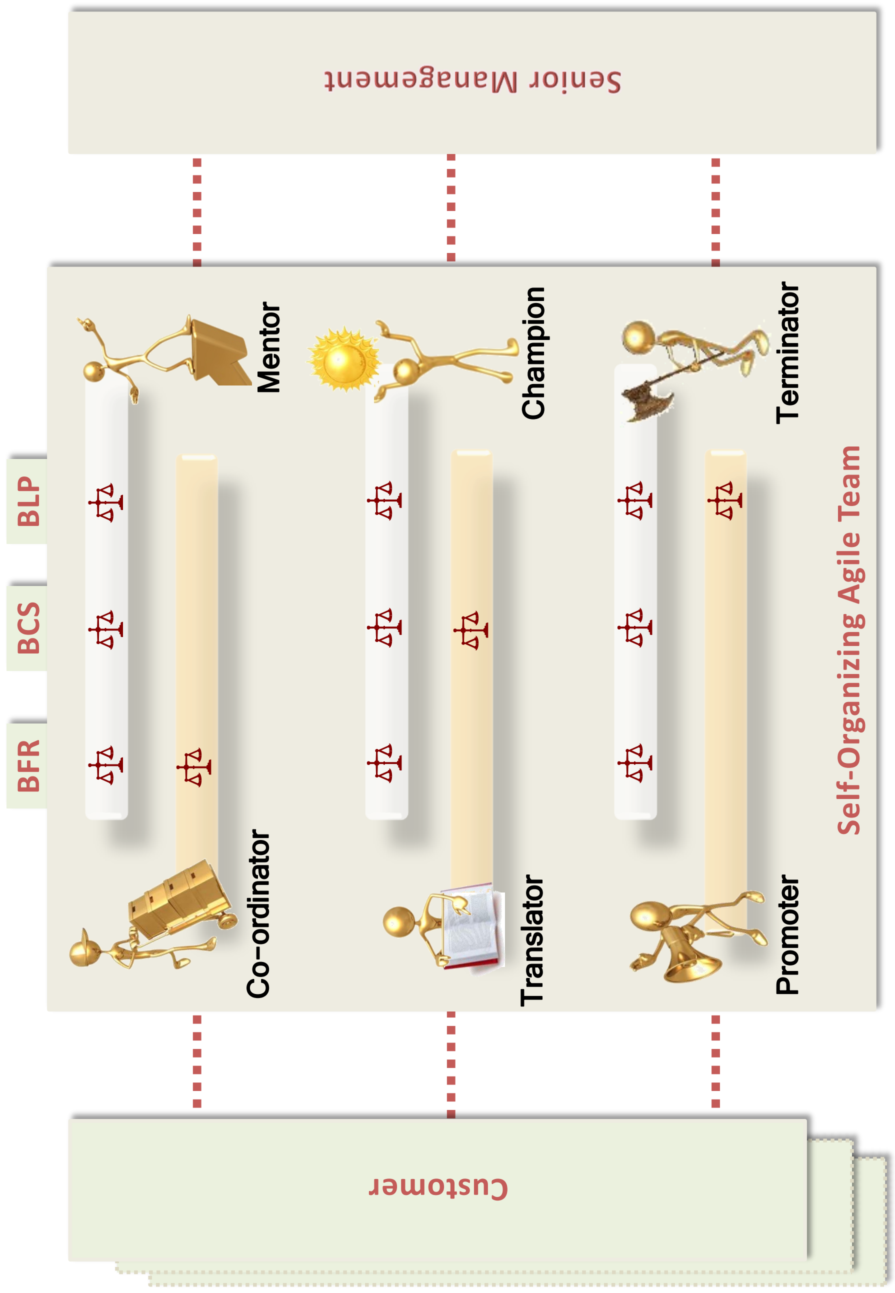


Figure 7.1: Theory of Self-Organizing Agile Teams. (Roles: Mentor, Co-ordinator, Translator, Champion, Promoter, and Terminator. Practices: Balancing Freedom and Responsibility (BFR); Balancing Cross-Functionality and Specialization (BCS); Balancing Learning and Iteration Pressure(BLP). Factors: Senior Management Support and Level of Customer Involvement.)

7.1.2 Role of the Agile Coach

The Agile coach initially plays most of the self-organizing Agile team roles. Over time, the self-organizing team roles will be taken up by the team members themselves. In more mature Agile teams, most members of the team have the caliber and experience to play any of the roles. For example, in mature teams, the *Mentor* role is generally played by experienced team members that help mentor newcomers on the team; the *Co-ordinator* and *Translator* roles are played by most members of the team as they gain experience in collaborating directly and frequently with their customers; the *Champion* and *Promoter* roles are played, as required, by more experienced members of the team. The *Terminator* role is played by the Agile coach with support from the team as they provide their input into the suitability of an individual to join or remain in an Agile team.

7.1.3 Self-Organizing Agile Team Practices

Self-organizing Agile teams balance *freedom and responsibility*, *cross-functionality and specialization*, and *continuous learning and iteration pressure.* These balancing acts affect how the team performs an integrated set of Agile practices:

- Balancing freedom and responsibility involves practices such as collective decision making through collective estimation and planning; collectively deciding teams and principles; and self-committing to team goals; self-assignment using story boards; self-monitoring through daily standup meetings and use of information radiators.
- Balancing cross-functionality and specialization involves practices such as multiple perspectives, group programming, and rotation.
- Balancing continuous learning and iteration pressure involves practices such as retrospective, learning spike, and pair-in-need.

These practices are performed in order to achieve and sustain the three fundamental conditions of self-organization: autonomy, cross-fertilization, and self-transcendence [154]; and to adhere to the general principles of self-organization: minimum critical specification, requisite variety, redundancy of functions, and learning to learn [115]. Detailed descriptions of the practices are provided in chapter 5.

7.1.4 Factors Influencing Self-Organizing Agile Teams

Self-organizing Agile teams face critical environmental factors that influence them: *senior management support* and *level of customer involvement*.

- Senior management within the development team's organization influences organizational culture, negotiating contracts, financial sponsorship, and human resource management, all of which impact the team. Self-organizing Agile teams attempt to secure senior management support through a *Champion* that highlights the benefits of Agile software development in terms of the business drivers that motivate business decisions. These business drivers include: applicability to project context, time-to-market, customer demands, and process improvement.

- Level of customer involvement also critically influences self-organizing Agile teams. Customer involvement influences the self-organizing Agile teams when gathering and clarifying requirements, prioritizing requirements, and securing customer feedback. Teams attempt to secure and maintain customer involvement through a *Promoter* that tries to convince the customers to collaborate, a *Co-ordinator* that helps them co-ordinate customer collaboration (in the face of inadequate customer involvement), and a *Translator* that helps translate between business and technical languages in an effort to improve communication. In the face of inadequate customer involvement, teams practice *Agile Undercover* strategies that include changing priority, story owners, just demos, e-collaboration, and extreme undercover [82, 74].

Detailed descriptions of these two critical factors influencing self-organizing Agile teams are provided in chapter 6.

This thesis also presents a description of the Grounded Theory method, examples of its application, and reflections on the challenges faced in using GT and strategies for overcoming them (chapter 3). Finally, this research has resulted in a number of publications focusing on various aspects of the theory of self-organizing Agile teams (Appendix A).

7.1.5 Roles-Practices-Factors Relationships

Figure 7.1 captures the relationships between the key contributions of this thesis: roles, practices, and factors. Members in Agile teams take on informal self-organizing roles in response to various challenges. Some of the roles specifically emerge in response to the two critical environmental factors. For example, the *Champion* role emerges to gain senior management support, the *Promoter* role emerges to secure customer involvement, the *Co-ordinator* role emerges to co-ordinate customer collaboration in case of inadequate customer involvement, and the *Translator* roles emerges to help translate between technical language used by the team and business language used by their customers. The other two roles, *Mentor* and *Terminator* emerge with support from senior management to help the team learn and practice Agile software development and remove members that are unable to adjust to the Agile way of working.

The self-organizing Agile team practices are supported by the roles and influenced by the environmental factors. For example, all three balancing acts and their underlying practices, require a *Champion* to convince senior management to support the practices, a *Mentor* to help guide the team through these practices, and a *Terminator* to identify and remove members that threaten the team by not being able to perform these practices.

In addition, balancing freedom and responsibility involves practices of collective estimation and planning and self-committing to a team goal, and require a *Co-ordinator* to gather and clarify requirements during estimation and planning in case of inadequate customer involvement. Balancing cross-functionality and specialization involves practices of group programming, rotation, and multiple perspectives, and lead to the emergence and strengthening of a *Translator* role. Balancing continuous learning and iteration pressure includes the practice of a learning spike that requires a *Promoter* to manage customer expectations in a way that allows the team to maintain a healthy team velocity while allowing for time to learn and up-skill themselves.

7.2 Limitations

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.

Data derived from interviews is known to be prone to bias [126]. There are four types of data that can be presented to the researcher: (a) Baseline data, the best description a participant can offer (b) Properline data, what the participant thinks it is proper to tell the researcher (c) Interpreted, what is told by a trained professional who wants to make sure that others see the data his professional way (d) Vaguing it out, the vague information provided by a participant that is not bothered to provide information to the researcher [58]. The researcher can encounter any of these. Software Engineering researchers may not be well trained in the art of interviewing for research and as such may struggle to illicit useful data from the participants. It takes time to build the ability to discern the type of data being provided during an interview and skill to be able to ask questions that can counter-check the data provided. Conducting semi-structured interviews with open-ended questions allows the researcher to ask the participants for specific and detailed examples. Semi-structured interviews also help to ask a question in multiple ways at different points in the interview.

Another effective way to ensure authenticity of the data collected through interviews and to validate the interpretation of the interview data, is to supplement it with observations of workplaces and activities [126]. The data derived from observations did not contradict, but rather supported our interview data, thereby strengthening it. A rounded perspective of the issues was gathered by interviewing practitioners representing other aspects of software development such as customer representative and senior management besides focusing on the development team (developer, tester, Agile coach, business analyst). In order to minimize any loss or misinterpretation, all data was personally collected and analyzed by the doctoral candidate—the author of this thesis.

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 [59]. A key contribution of a GT study, carried out correctly, is that it focuses on conceptualization and produces flexible, modifiable concepts with "*immense grab*" [61]. These concepts inter-relate 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 and other disciplines will help generate a more generalized theory.

7.3 Discussion

The grounded theory of self-organizing Agile teams presented in this thesis is a first large-scale study of this topic in the field of Agile software development.

The various aspects of the theory build upon previous work. Sections 4.8, 5.7, and 6.5 discuss each of the main contributions of this thesis in the context of related work. This section summarizes those discussions.

Team Roles Team roles have been described outside the field of software development [9, 22, 134]. Belbin suggested nine team roles based on individual behaviours traits of team members. In contrast, the self-organizing Agile team roles are focused on facilitating self-organization. A co-ordinator in Belbin's team roles theory focuses on team objectives and delegates work. The *Co-ordinator* role identified in our research, on the other hand, helps co-ordinate between the team and their customers and does not delegate work.

Ancona and Caldwell and Sawyer et al. describe five boundary-spanning roles focused on encouraging communication of the team with external stake-holders [9, 134]. An ambassador role in their study represents the team to external stake-holders 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. The boundary spanning roles also consist of a co-ordinator which is similar to our *Co-ordinator* role, focusing on communication with external groups while keeping them informed of the team's progress.

The self-organizing Agile team roles identified in this research include internal, external, and interfacing roles. The *Mentor* and *Terminator* roles are primarily internal facing, the *Champion* and *Promoter* are external facing, and the *Co-ordinator* and *Translator* roles are interfacing roles between the team and their external stake-holders (senior management and customers). The self-organizing nature of these roles identified in this research is further consolidated when compared to the characteristics of self-organizing teams defined by Anderson and McMillan [10]. Detailed discussion of team roles in relation to relevant literature on team roles has been provided in section 4.8.

Role of the Agile Coach Section 4.7 describes the role of the Agile coach as presented in literature. In particular, a change in management style from command-and-control to leadership and collaboration has been predicted [16, 119, 40, 148]. There has been no substantial research exploring the role of the Agile coach across multiple organizations and countries however. Our theory helps describe the role of Agile coach in terms of the self-organizing Agile roles they are likely to play in a self-organizing Agile team.

Team Practices Studies describe mature Agile teams as highly collaborative and self-organizing in nature, exhibiting responsibility on both individual and team levels [142]. These studies emphasize the importance of story boards in collaborative activities of mature Agile teams [141]. Our research confirms that status report meetings and information radiators used as self-monitoring practices by Agile teams enable them to balance freedom and responsibility effectively.

Self-monitoring practices have been shown to influence responsibility and ownership in Agile teams [141, 142, 162]. Daily standups and the use of information radiators have been found to increase social answerability and awareness in Agile teams [162].

Moe et al. explored the teamwork challenges that arise when introducing a self-organizing Agile team [112]. The results indicate that the main challenges in achieving team effectiveness include problems with highly specialized skills and the corresponding division of work. Our research confirms their findings that Agile teams need to balance cross-functionality and specialization in order to sustain self-organization. Furthermore, our research provides guidance on concrete practices that enable teams to achieve this balance: multiple perspectives, group programming, and rotation.

Pairing has been described as a mechanism for learning through conversations between pairs [131, 164, 165]. Studies have acknowledged that pair programming can be exhausting [131, 51, 165]. Our research found that teams practice pair-in-need instead of compulsory, consistent pairing. Teams found pair-in-need to be a useful way to achieve learning while managing the pressures of delivering team goals (5.3). Detailed discussion of the self-organizing Agile team practices in relation to relevant literature on team roles has been provided in section 5.7.

Environmental Factors The importance of senior management support in adoption of Agile methods has been widely acknowledged [19, 35, 44, 51, 67, 119, 148, 157]. Additionally, our research shows that senior management support is imperative for the sustenance of self-organizing Agile teams (section 6.1). Our study confirms that senior management will need to change several organizational processes in order to make them conducive for self-organizing Agile teams, such as changing their organizational culture [51, 119]. As Boehm suggests, these changes may be non-trivial [29]. The extent of changes required will depend on how far the current environment is from an ideal environment for self-organizing Agile teams [157].

Customer collaboration is a vital feature in Agile software development [35, 73, 72, 90, 100, 107, 110]. Our grounded theory establishes customer involvement as a critical environmental factor that influences self-organization in Agile teams (6.3).

The role of the customer in XP has been described at length as a grounded theory by Martin [107]. Martin identified several roles that form an informal customer team, of which the Negotiator role is the closest to the on-site customer described in XP. Our theory identifies a similar *Co-ordinator* role on the development team side who is responsible for collaborating with the Negotiator on the customer side.

An ideal customer representative is an individual who has both thorough understanding of, and ability to, express the project requirements and the authority to take strategic decisions [44, 55, 66, 119]. This representative must be CRACK (Collaborative, Responsible, Authorized, Committed, Knowledgeable) [29]. Additionally, our study suggests that the customer representative should understand the basics of Agile methods and the theory of self-organizing Agile teams.

A gap between ideal and real levels of customer involvement on Agile projects has been acknowledged [31, 42, 44, 90, 103, 127, 131]. Studies have reported the practice of using proxy or surrogate customers in the face of inadequate customer involvement [31, 66, 87, 101, 103]. Our study identified the *Co-ordinator* role which acted as a team representative, co-ordinating collaboration with customers (section 4.2).

Detailed discussion of these factors influencing self-organizing Agile teams in relation to relevant literature has been provided in section 6.5.

Evaluating the Grounded Theory As per Glaser's recommendation, a grounded theory can be evaluated on the basis of four criteria: fit, work, relevance, and modifiability [59] (section 3.4.10). This section evaluates the theory of self-organizing Agile teams against these criteria:

Fit: 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 email 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: 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 NZ 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: 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 experts at major international conferences to gain their feedback [12, 74, 78]. 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 [69, 89, 84, 49, 102].

Modifiability: 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 (chapter 4).

In addition to these criteria, the ability of a theory to fit and extend 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, literature becomes an important source of validating the emerging theory. For example, once we had established the three balancing acts as the practices of Agile teams that particularly enable self-organization, we conducted an extensive literature review on self-organization in and outside software engineering. We found that previous literature in organizational theory had defined the general principles of self-organization [115]. Literature in Agile software development described the three conditions of self-organization [154]. Both these principles and conditions of self-organization fit perfectly with our practices of self-organizing teams. All the main categories derived from this GT study have been compared to existing literature and presented in sections 4.8, 5.5, 5.6, 5.7, 6.5.1, and 6.5.2.

7.4 Implications for Practice

Our theory of self-organizing Agile teams has several implications for practitioners. The following sections present the implications of this theory for teams, their Agile coaches, senior management, and customers.

7.4.1 Implications for Teams

The transition of becoming a self-organizing Agile team is not easy. The roles and practices described in this thesis 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 standups 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 *Co-ordinator* 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, most 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*.

7.4.2 Implications for Agile Coaches

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 themselves 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 thesis should help Agile coaches better understand the responsibilities they are likely to take on at the different stages of the team's maturation. The practices described in this thesis should assist Agile coaches in guiding their team into self-organization. The critical factors identified in this thesis should help Agile coaches know what to expect in terms of challenges and how to react through the roles and practices. An important implication for the Agile coach, however, is to always be mindful of the self-organizing nature of these roles and practices and facilitate their emergence rather than forcing them on the team.

7.4.3 Implications for Senior Management

Senior management must be made aware of their influence on the ability of Agile teams to self-organize. An important aspect of this awareness is understanding both Agile methods, and their role in creating a conducive environment for Agile teams to achieve and sustain self-organization: an organizational culture which is characterized by trust, openness, free flow of information, and informality.

Senior management must decide whether such changes are beneficial for their organization. The business drivers discussed in section 6.2 should help guide senior management in making this decision. Senior management can try to assess the advantages they stand to gain in making these changes to accommodate self-organizing Agile teams. Examples of organizations that will likely benefit from self-organizing Agile teams include those that cater to product/applications that require frequent changes and innovation. Some senior managers may find that the effort involved in undertaking such changes outweigh the benefits of introducing Agile methods, especially when they mostly cater product/applications that are design and architecture-driven, safety-critical, or have a slow rate of change in requirements [75].

7.4.4 Implications for Customers

Customers should be made aware of their influence on the self-organizing ability of Agile teams. Customers will need to carefully select members from within their organization as representatives to collaborate with the development teams. Where possible, such as in the case of an in-house customer, the self-organizing Agile team should be consulted when selecting a representative. The representative should be provided enough time and authority to effectively collaborate with the team.

Customers should try to understand their role when starting an Agile project. To this end, vendor organizations may consider offering relevant training to their new customers. Customers should attempt to bridge the gap between ideal and real levels of involvement and collaboration with self-organizing Agile teams as it ultimately benefits their project.

7.5 Future Work

7.5.1 Stages of Becoming a Self-Organizing Agile Team

This research suggests a preliminary model of becoming a self-organizing Agile team which involves 3 stages: *establishing*, *practicing*, *transcending*.

Establishing The establishing stage of becoming a self-organizing Agile team is where a group of software practitioners come together to form a team. In a non-Agile organization, this may be the first self-organizing Agile team—a pilot team. A clear indicator of this stage is a lack of knowledge about Agile principles, values, and practices among the team members. The presence of a *Mentor* in the form of an Agile coach is extremely important in the initial stages, as the *Mentor* familiarizes the team with Agile principles, values, and practices and guides them through the first few iterations (section 4.1).

Many problems, such as people-related issues, are likely to surface in this stage. Some team members may become anxious about the new environment of working and their own roles in the team. Individuals who are not comfortable working in an open Agile environment show signs of distress or aggression. A *Terminator* or *Mentor* can try to convince them to change their mindsets, otherwise, seek senior management support in removing such individuals from the team.

Practicing In the practicing stage, the team is expected to be familiar with the fundamentals of Agile software development and be comfortable with most basic practices. The team should feel more confident about their abilities to work in an Agile environment. Team members should experience high enthusiasm, energy, cohesion, and motivation in this stage. The team starts to devise strategies to overcome the challenges posed by the environmental factors, such as level of customer involvement.

Transcending Few teams will reach this stage, depending on their internal team development and strong support of the two critical environmental factors—senior management support and level of customer involvement. In this stage, the self-organizing Agile team roles should become dormant at the team level, with most of the challenges they address being resolved. Team members will likely experience high performance, morale, and general team spirit and feel very positive about themselves, their project, their management, and their customers. A distinct team culture is expected to emerge by this stage.

The research also suggests two extended stages—*Propagating*: where a self-organizing Agile team leads to propagation of more teams in the organization, and *Terminating*—where a self-organizing Agile team is disintegrated for various reasons and there is no further propagation across the organization. This preliminary model of becoming a self-organizing Agile team suggested by this research is similar to a popular small group formation model—norming, forming, storming, and performing—suggested by Tuckman in 1965 [160]. This preliminary model also supports the *Shu-Ha-Ri* stages of mastery as applied to Agile software development [37].

Future work could explore the stages suggested in this model on new and mature Agile teams, such as a detailed study tracing the progress of teams from the initial to the advanced stages of self-organization.

7.5.2 Scaling Self-Organization: From Self-Organizing Teams to Self-Organizing Organizations

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 co-ordinating customers collaboration, and removing cultural misfits become organization-wide concerns. In response, the self-organizational team roles of *Mentor*, *Co-ordinator*, *Translator*, *Champion*, *Promoter*, and *Terminator* can be mirrored at the organizational level.

The presence of these organization-wide roles was indicated in two mature Agile organizations towards 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.

7.5.3 Exploring Cultural Implications

Our cross-cultural research looked at Agile practitioners from New Zealand and India but did not find any notable co-relations between the teams' national cultures and the main components of our theory. In other words, the self-organizing roles, practices, and factors were consistent across the two national cultures. There was, however, some indication of the existence of a distinct Agile team culture reflected by practices of self-assignment, group programming, collective decision making, daily standup, using information radiators, retrospectives, and pair-in-need. Researchers such as Sharp et al. and Whitworth et al. have classified Agile team culture in similar ways [141, 162]. Future studies could explore in more detail any cultural implications of our theory in different contexts.

7.5.4 Diagnostic Tools

Our study describes the changes senior management needs to make to support self-organizing Agile teams, as well as the motivators (business drivers) that drive senior management's business decisions (sections 6.1 and 6.2). Managers need to compare the extent of changes required in the organization with the likely benefits from introducing these teams.

Future studies could use these guidelines to build diagnostic tools to help senior management evaluate the expected benefit from supporting self-organizing Agile teams.

Appendices

Appendix A: List of Publications

1. Hoda, R, Noble, J, Marshall S. *Developing a Grounded Theory to Explain the Practices of Self-Organizing Agile Teams.* Empirical Software Engineering Journal (In Press) 2011

2. Hoda, R, Noble, J, Marshall S. *The Impact of Inadequate Customer Involvement on Self-Organizing Agile Teams.* Journal of Information and Software Technology, Vol. 53, 521-534, May 2011

3. Rashina Hoda, James Noble, Stuart Marshall. *Supporting Self-Organizing Agile Teams: What's Senior Management Got To Do With It?* XP2011, Madrid, Spain, May 2011 [To Appear]

4. Hoda, R, Noble, J, Marshall S. *Organizing Self-Organizing Agile Teams.* ICSE, Cape Town, South Africa, 2010

5. Hoda, R, Noble, J, Marshall S. *Balancing Acts: Walking the Agile Tightrope.* CHASE workshop at ICSE, Cape Town, South Africa, 2010

6. Hoda, R, Kruchten, P, Noble, J, Marshall S. *Agility in Context.* OOPSLA, Reno/Nevada, USA, 2010

7. Hoda, R, Noble, J, Marshall S. *Using Grounded Theory to Study the Human Aspects of Software Engineering.* HAoSE workshop at SPLASH, Reno/Nevada, USA, 2010

8. Hoda, R, Noble, J, Marshall S. *Agile Undercover: When Customers Don't Collaborate.* XP2010, Trondheim, Norway, 2010

9. Hoda, R, Noble, J, Marshall S. *What Language Does Agile Speak?.* XP2010, Trondheim, Norway, 2010

10. Hoda, R, Noble, J, Marshall S. *How Much is Just Enough: Some Documentation Patterns on Agile Projects.* EuroPLoP, Germany, 2010

11. Hoda, R, Noble, J, Marshall S. *Negotiating Contracts for Agile Projects: A Practical Perspective.* XP2009, Sardinia, Italy, 2009

12. Hoda, R, Noble, J, Marshall S. *Don't Mention the 'A' Word: Agile Undercover.* Research-in-Progress workshop at Agile2009, Chicago, USA, 2009

13. Hoda, R, Noble, J, Marshall S. *Agile Project Management: A Grounded Theory Perspective.* NZCSRSC, Auckland, New Zealand, 2009

14. Hoda, R, Noble, J, Marshall S. *Exploring the Role of the Manager in Agile Projects.* ACDC, Wellington, New Zealand, 2009

15. Hoda, R, Noble, J, Marshall S. *A for Agile, Issues with Awareness and Adoption.* Research-in-Progress workshop at Agile2008, Toronto, Canada, 2008

16. Hoda, R, Noble, J, Marshall S. *Agile Project Management.* NZCSRSC, Christchurch, New Zealand, 2008

Appendix B: Approved HEC Application and Documents

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

HUMAN ETHICS COMMITTEE
Application for Approval of Research Projects

Please write legibly or type if possible. **Applications must be signed by supervisor (for student projects) and Head of School**

Note: The Human Ethics Committee attempts to have all applications approved within three weeks but a longer period may be necessary if applications require substantial revision.

1 NATURE OF PROPOSED RESEARCH:

(a) Staff Research ☐	Student Research ☒	(tick one)
(b) If Student Research	Degree PhD	Course Code **COMP 690**
(c) Project Title: Agile Project Management		

2 INVESTIGATORS:

(a) Principal Investigator	
Name	Rashina Hoda
e-mail address	rashina@gmail.com
School/Dept/Group	Computer Science

(b) Other Researchers Name	Position
None	

(c) Supervisor (in the case of student research projects)	
Dr. James Noble and Dr. Stuart Marshall	

3 DURATION OF RESEARCH

(a) Proposed starting date for data collection 1st January 2008
(Note: that NO part of the research requiring ethical approval may commence prior to approval being given)

(b) Proposed date of completion of project as a whole 1st May 2010

4 PROPOSED SOURCE/S OF FUNDING AND OTHER ETHICAL CONSIDERATIONS

(a) Sources of funding for the project

Please indicate any ethical issues or conflicts of interest that may arise because of sources of funding e.g. restrictions on publication of results

None

(b) Is any professional code of ethics to be followed **Y** ☐ **N** ☒

If yes, **name**

(c) Is ethical approval required from any other body **Y** ☐ **N** ☒

If yes, name and indicate when/if approval will be given

Not applicable

5 DETAILS OF PROJECT

Briefly Outline:

(a) The objectives of the project

To explore the concept of Agile Project Management within companies/practitioners/mentors using Agile software methodologies such as extreme programming (XP), Scrum, Crystal, etc.
In particular the research will use qualitative research methods like grounded theory to derive important conculsions about the practices of project management within the Agile software development field in New Zealand, India and possibly other international markets.
The investigation is expected to delve into the following sub-topics, and explore:

- the role of the project manager in an Agile project
- the process and problems of transitioning into an Agile company/practitioner
- the issues around offshoring or outsourcing of Agile software projects.

b) Method of data collection

A combination of qualitative methods will be used for data collection based on qualitative research methods such as grounded theory. These include semi-structed interviews, surveys, questionnaires, observations, etc. In case of a project being followed through, effort will be made to conduct interviews or have questionnaires filled out at important milestones of the project or at mutually agreed regular intervals.

(c) The benefits and scientific value of the project

The research will explore the use of Agile methodologies in real life companies/practitioners to gain broader and in-depth understanding of the process. It will aim to formulate the best practices and experiences of Agile project management as witnessed by different companies/practitioners.
- It will allow Agile project managers to gain better understanding of their role in Agile projects management issues as explored in this research in various companies.
- By exploring the experiences of companies transitioning into the Agile software development, the research will help new companies/practitioners to adopt tried and tested ways of successful transitioning.
- Similarly, the research will allow companies to better understand the issues around offshoring or outsourcing of Agile projects.

(d) Characteristics of the participants

Software project managers, developers, team leaders, practitioners, authors, mentors, or consultants involved with agile methodologies.

(e) Method of recruitment

Companies and practitioners will be approached via email, Agile community lists, and user groups with a brief outline of the intended research.
The researcher will then collaborate with individual organisations or practitioners to clearly define mutually agreeable terms. This will include the purpose and scope of the project, the collection and use of data, the time commitments expected from the staff, and feedback procedures.
Although there's no rigid limit on the number of participants, we estimate that we may need to interview up to a maximum 20 participants.
An initial draft of this proposal is attached to this application with supplementary information sheet, interview guide and consent form.

(f) Payments that are to be made/expenses to be reimbursed to participants

None

(g) Other assistance (e.g. meals, transport) that is to be given to participants

None

(h) Any special hazards and/or inconvenience (including deception) that participants will encounter

None

(i) State whether consent is for:

(i)	the collection of data	[✓ Y]	N
(ii)	attribution of opinions or information	Y	[✓ N]
(iii)	release of data to others	Y	[✓ N]
(iv)	use for a conference report or a publication	[✓ Y]	N

(v) use for some particular purpose (specify) Y☐ N☒

Not applicable

Attach a copy of any questionnaire or interview schedule to the application

(j) How is informed consent to be obtained (see sections 4.1, 4.5(d) and 4.8(g) of the Human Ethics Policy)

(i) the research is strictly anonymous, an information sheet is supplied and informed consent is implied by voluntary participation in filling out a questionnaire for example (include a copy of the information sheet) Y☐ N☒

(ii) the research is not anonymous but is confidential and informed consent will be obtained through a signed consent form (include a copy of the consent form and information sheet) Y☒ N☐

(iii) the research is neither anonymous or confidential and informed consent will be obtained through a signed consent form (include a copy of the consent form and information sheet) Y☐ N☒

(iv) informed consent will be obtained by some other method (please specify and provide details) Y☐ N☒

Not applicable

With the exception of anonymous research as in (i), if it is proposed that written consent will not be obtained, please explain why

Not applicable

(k) If the research will not be conducted on a strictly anonymous basis state how issues of confidentiality of participants are to be ensured if this is intended. (See section 4..1(e) of the Human Ethics Policy). (e.g. who will listen to tapes, see questionnaires or have access to data). Please ensure that you distinguish clearly between anonymity and confidentiality. Indicate which of these are applicable.

(i) access to the research data will be restricted to the investigator Y☐ N☒

(ii) access to the research data will be restricted to the investigator and their supervisor (student research) Y☒ N☐

(iii) all opinions and data will be reported in aggregated form in such a way that individual persons or organisations are not identifiable Y☒ N☐

(iv) Other (please specify)

Not applicable

(l) Procedure for the storage of, access to and disposal of data, both during and at the conclusion of the research. (see section 4.12 of the Human Ethics Policy). Indicate which are applicable:

(i) all written material (questionnaires, interview notes, etc) will be kept in a locked file and access is restricted to the investigator **Y** ☒ **N** ☐

(ii) all electronic information will be kept in a password-protected file and access will be restricted to the investigator **Y** ☒ **N** ☐

(iii) all questionnaires, interview notes and similar materials will be destroyed:
(a) at the conclusion of the research **Y** ☐ **N** ☒

<u>**or**</u> (b) 3 years after the conclusion of the research **Y** ☒ **N** ☐

(iv) any audio or video recordings will be returned to participants and/or electronically wiped **Y** ☒ **N** ☐

(v) other procedures (please specify):

None

If data and material are not to be destroyed please indicate why and the procedures envisaged for ongoing storage and security

Not applicable

(m) Feedback procedures (See section 7 of Appendix 1 of the Human Ethics Policy). You should indicate whether feedback will be provided to participants and in what form. If feedback will not be given, indicate the reasons why.

Participants will be kept updated about research results and all important findings. They will also be given feedback and allowed to review the interpretation of their comments/data.

(n) Reporting and publication of results. Please indicate which of the following are appropriate. The proposed form of publications should be indicated on the information sheet and/or consent form.

(i) publication in academic or professional journals **Y** ☒ **N** ☐

(ii) dissemination at academic or professional conferences **[✓ Y]** **N** ☐

(iii) deposit of the research paper or thesis in the University Library (student research) **[✓ Y]** **N** ☐

(iv) other (please specify)

None

Signature of investigators as listed on page 1 **(including supervisors) and Head of School.**

NB: <u>All investigators and the Head of School must sign before an application is submitted for approval</u>

		Date	
		Date	
		Date	

Head of School:

		Date	

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Agile Project Management - Information Sheet

General Information

This research is being conducted as a part of studies towards a PhD degree in the department of Computer Science at Victoria University of Wellington, New Zealand.

Student:	Rashina Hoda (hodarash@mcs.vuw.ac.nz, +64 4 463 6778)
Supervisors:	Dr. James Noble (kjx@mcs.vuw.ac.nz, +64 4 463 6736)
	Dr. Stuart Marshall (stuart.marshall@vuw.ac.nz, +64 4 463 6730)
Research Topic:	Agile Project Management

Aim of the Research

The objective of this research is to explore the concept of Agile Project Management within companies/practitioners/mentors using Agile software methodologies such as XP, Scrum, Crystal. The investigation is expected to delve into the following sub-topics, and explore:

- the role of the project manager in an Agile project
- the process and problems of transitioning into an Agile company/practitioner
- management of offshored or outsourced Agile software projects.

Method of Research and Interviews

The research will use qualitative analysis methods to gather valuable data regarding various issues in Agile Project Management in New Zealand and India. We have sought and have been granted approval by the Human Ethics Committee to conduct these interviews and observations.

In order to gather information regarding the topic, interviews will be conducted to gain insight and data from project managers, developers, practitioners, mentors, and consultants who have practical experience in the field of Agile project management. The data collected in the form of interview transcripts or project results will be treated as strictly confidential (please see details under 'Confidentiality' section below.) We would ideally like to conduct 2 or 3 interviews at different important stages of the project. Each interview would last for roughly an hour and will be held at the interviewee's workplace or as mutually agreed between the researcher and the interviewee. The interviews will be taped to reduce the risk of interviewer not being able to note down all information provided by interviewee. An interview guide is attached herewith.

Purpose of Data Collection

The data collected will be analysed carefully to derive important conclusions about the practices of project management within the Agile software development field. Papers may be published in journals and conferences during the course of the research for the benefit of the larger research community. The final thesis report will be published as a PhD thesis and will be held at the Victoria University Library.

Confidentiality and Consent

All materials collected will be stored in a confidential way and will be destroyed at the completion of the research. No personal information or details will be collected during the interview. The data collected will be kept confidential to the researcher (myself), and my supervisors Dr. James Noble and Dr. Stuart Marshall. The thesis report and any papers published as a result of the study will not

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Consent for Participation in Research

Topic of Research: Agile Project Management
Researcher: Rashina Hoda, Victoria University of Wellington, New Zealand

I have been provided with and have understood the information regarding this research and the confidentiality conditions. I have been given the opportunity to ask questions and have them answered to my satisfaction.

I agree to be interviewed by Rashina Hoda for the purpose of this research contributing towards her PhD degree and resultant thesis and conference papers publications. I also understand that I may withdraw from this research upto 30 days after the data collection/interview.

I give my consent to the collection and use of my opinions, perceptions, information and experiences during this research.

I agree to have the interviews sound-recorded (to reduce the risk of interviewer not being able to note down all information provided by interviewee)?

YES NO

I would like to receive a copy of any publications that are based on these interviews?

YES NO

If yes, please provide an email or mailing address below.

____

____

Name: ______________________

Signed:______________________

Date: ______________________

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Agile Project Management – Interview Guide

General Information

Interview Date: ___________________________

Interview Venue: ___________________________

____

Topic: Agile Project Management

We will discuss any or all of the following depending on whats relevant and applicable to the interviewee's experience.

- role of project manager in Agile projects
- process and problems of transitioning into an Agile framework
- management of outsourced or off-shored Agile projects

Agenda

	Category	**Duration**
1.	Explain topic, agenda, and rules of interview	5 mins
2.	Previous experience with Agile methodologies, Agile project management, transitioning, and outsourcing	10mins
3.	Depending on interviewee's experience: - define your role and responsibilities as project manager - details of transitioning into an Agile framework - detailed setup of outsourced Agile projects	15 mins
4.	Discuss things that worked well for the project (your idea of best practices) with respect to any or all of the above points (refer 3)	10 mins
5.	Discuss problems and issues with respect to any or all of the above (3)	10 mins
6.	Suggest improvements on any or all of the above areas (3)	5 mins
7.	Closing (fix next interview session where applicable, explain feedback process to interviewee.)	5 mins

Rules of Interview

- Interviews will be taped, upon mutual agreement, to reduce the risk of interviewer not being able to note down all information provided by the interviewee.
- Interviewees can be provided with interpretations of their comments/data collected during the interview, if required by the interviewee.
- Interviewee will be allowed to discuss any other relevant issue not covered by the interview agenda.

Interview Questions

1. How did you learn about Agile?

2. Is there a live Agile project that you are working on?

3. What is the project about (what flavour of Agile are you using)?

4. What's the team size and project duration?

5. Was the customer tuned into Agile or did you promote it?

6. What's your role and responsibilities in the project?

7. What difficulties have you faced so far on this project?

8. What are the main issues faced by you (as a manager/leader) when dealing with:
   A. customers

   B. internal team and management

9. At this point, what are your expectations of the project (how long will it take, on budget/ on time)?

10. In your wider experience, what are the advantages of Agile project management?

11. Disadvantages, if any?

12. Please describe your experience of transitioning into an Agile framework (share particular project experience)

13. what were some of the biggest obstacles in transitioning and how did you get around them?

14. who best supported the process?

15. what went wrong?

16. what would you advice companies thinking of transitioning into Agile?

17. In your opinion, whats the best way to promote Agile?

18. Is there anything else that you feel we should have discussed?

Bibliography

[1] Book reviews comptes rendus. *Canadian Public Administration 32*, 2 (1989), 320–339.

[2] *A Guide To The Project Management Body Of Knowledge (PMBOK Guides).* Project Management Institute, 2004.

[3] *Comparing PMBOK and Agile Project Management Software Development Processes.* Springer, 2007, pp. 378–383.

[4] Abraham, L. Cultural differences in software engineering. In *ISEC '09* (New York, 2009), ACM, pp. 95–100.

[5] Abrahamsson, P. *Agile Software Development Methods: Review and Analysis (VTT publications).* VTT publications, 2002.

[6] Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J. New directions on Agile methods: a comparative analysis. In *Proceedings of 25th International Conference on Software Engineering* (2003), pp. 244–254.

[7] Allan, G. A critique of using grounded theory as a research method. *EJBRM 2*, 1 (2003).

[8] Allan, G. The legitimacy of grounded theory. *Key Note Address 5th European Conference on Research Methodology for Business and Management* (2006), 1–8.

[9] Ancona, D. G., and Caldwell, D. F. Beyond task and maintenance: Defining external functions in groups. *Group Organization Management 13*, 4 (1988), 468–494.

[10] Anderson, and McMillan. Of ants and men: self-organized teams in human and insect organizations. *Emergence: Complexity Organization 5*, 2 (2003), 29–41.

[11] Anderson, L., Alleman, G., Beck, K., Blotner, J., Cunningham, W., Poppendieck, M., and Wirfs-Brock, R. Agile management - an oxymoron?: who needs managers anyway? In *OOPSLA '03* (New York, 2003), ACM, pp. 275–277.

[12] APN. Agile professionals network. World Wide Web electronic publication, `http://www.agileprofessionals.net/`, last accessed on 20th Sep 2010.

[13] ASCI. Agile software community of india. World Wide Web electronic publication, `http://www.agileindia.org/`, last accessed on 20th Sep 2010.

[14] Ashby, R. *An introduction to cybernetics.* Chapman and Hall, London, 1956.

[15] Aston, J., Laroche, L., and Meszaros, G. Cowboys and indians: Impacts of cultural diversity on Agile teams. In *Proceedings of the conference on Agile 2008* (Toronto, Canada, 2008), IEEE Computer Society, pp. 423–428.

[16] Augustine, S. *Managing Agile Projects.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[17] Baker, J. Tightening the iron cage: Concertive control in self-managing teams. *Administrative Science Quarterly 38*, 3 (1993), 408–437.

[18] Beaumont, S. The definition of ready. Xebia Blogs, url = http://blog.xebia.com/2009/06/19/the-definition-of-ready, last accessed on 9th Nov 2010.

[19] Beck, K. *Extreme Programming Explained: Embrace Change*, first ed. Addison-Wesley Professional, 1999.

[20] Beck, K., and Andres, C. *Extreme Programming Explained: Embrace Change (2nd Edition).* Addison-Wesley Professional, 2004.

[21] Begel, A., and Nagappan, N. Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. In *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 255–264.

[22] Belbin, R. *Team roles at work.* Butterworth-Heinemann, 1993.

[23] Benoliel, J. Q. Grounded theory and nursing knowledge. *Qualitative Health Research 6*, 3 (1996), 406–428.

[24] Berteig, M. Team self-organization. Agile Advice, `http://www.agileadvice.com/archives/2005/12/agile_work_uses_2.html`, last accessed on 9th Nov 2010.

[25] Boehm, B. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes 11*, 4 (1986), 14–24.

[26] Boehm, B. Get ready for Agile methods, with care. *Computer 35*, 1 (Jan. 2002), 64 –69.

[27] Boehm, B. A view of 20th and 21st century software engineering. In *ICSE '06: Proceedings of the 28th international conference on Software engineering* (New York, NY, USA, 2006), ACM, pp. 12–29.

[28] Boehm, B., and Turner, R. Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Softw. 22*, 5 (2005), 30–39.

[29] Boehm, B. W., and Turner, R. Rebalancing your organization's agility and discipline. In *In XP/Agile Universe* (2003), pp. 1–8.

[30] Brooks, P. F. *Mythical Man-Month, Second Edition.* Addison-Welsey, 1995.

[31] Cao, L., Mohan, K., Xu, P., and Ramesh, B. A framework for adapting Agile development methodologies. *European Journal of Information Systems 18*, 4 (2009), 332–343.

[32] Carver, J. The impact of background and experience on software inspections. *Empirical Software Engineering 9*, 3 (2004), 259–262.

[33] Charmaz, K. *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis (Introducing Qualitative Methods series)*, 1 ed. Sage Publications Ltd, 2006.

[34] Chau, T., and Maurer, F. Knowledge Sharing in Agile Software Teams. In *Logic versus Approximation* (2004), pp. 173–183.

[35] Chow, T., and Cao, D. A survey study of critical success factors in Agile software projects. *Journal of Systems and Software 81*, 6 (2008), 961–971.

[36] Chua, W. F. Radical developments in accounting thoughts. *The Accounting Review 61*, 4 (1986), 601–632.

[37] Cockburn, A. *Agile software development.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[38] Cockburn, A. *People and Methodologies in Software Development.* PhD thesis, University of Oslo, Norway, 2003.

[39] Cockburn, A. *Crystal clear: a human-powered methodology for small teams.* Addison-Wesley Professional, 2004.

[40] Cockburn, A., and Highsmith, J. Agile software development: The people factor. *Computer 34*, 11 (2001), 131–133.

[41] Coleman, G., and OConnor, R. Using grounded theory to understand software process improvement: A study of Irish software product companies. *Inf. Softw. Technol. 49*, 6 (2007), 654–667.

[42] Conboy, K. Agility from first principles: Reconstructing the concept of agility in information systems development. *Info. Sys. Research 20*, 3 (2009), 329–354.

[43] Control Chaos. World Wide Web electronic publication, `http://www.controlchaos.com/old-site/rules.htm`, last accessed on 16th Sep 2010.

[44] Coram, M., and Bohner, S. The impact of Agile methods on software project management. In *Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 363–370.

[45] Crabtree, C. A., Seaman, C. B., and Norcio, A. F. Exploring language in software process elicitation: A grounded theory approach. In *ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 324–335.

[46] Creswell, J. W. *Research Design: qualitative, quantitative, and mixed methods and approaches (second edition).* Sage Publications, 2003.

[47] Dagenais, B., Ossher, H., Bellamy, R. K. E., Robillard, M. P., and de Vries, J. P. Moving into a new software project landscape. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (2010), ACM, pp. 275–284.

[48] Derby, E., and Larsen, D. *Agile Retrospectives: Making Good Teams Great.* Raleigh: Pragmatic Bookshelf, 2006.

[49] 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`, last accessed on 9th Nov 2010.

[50] Dickinson, T., and McIntyre, R. A conceptual framework of teamwork measurement. *Team Performance Assessment and Measurement: Theory, Methods, and Applications* (1997), 19–43.

[51] Dybå, T., and Dingsoyr, T. Empirical studies of Agile software development: A systematic review. *Inf. Softw. Technol. 50*, 9-10 (2008), 833–859.

[52] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. Selecting empirical methods for software engineering research. *Guide to Advanced Empirical Software Engineering* (2008), 285–311.

[53] Elliot, N., and Lazenbatt, A. How to recognize a 'quality' grounded theory research study. *Australian Journal of Advanced Nursing 22*, 3 (2005), 48–52.

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

[55] Fraser, S., Martin, A., Biddle, R., Hussman, D., Miller, G., Poppendieck, M., Rising, L., and Striebeck, M. The role of the customer in software development: the XP customer - fad or

fashion? In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (New York, NY, USA, 2004), ACM, pp. 148–150.

[56] Fraser, S., Reinitz, R., Eckstein, J., Kerievsky, J., Mee, R., and Poppendieck, M. Xtreme programming and Agile coaching. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (New York, NY, USA, 2003), ACM, pp. 265–267.

[57] Georgieva, S., and Allan, G. Best practices in project management through a grounded theory lens. *Electronic Journal of Business Research Methods 6*, 1 (2008), 43–52.

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

[59] Glaser, B. *Basics of Grounded Theory Analysis: Emergence vs Forcing.* Sociology Press, Mill Valley, CA, 1992.

[60] Glaser, B. *Doing Grounded Theory: Issues and Discussions.* Sociology Press, Mill Valley, CA, 1998.

[61] Glaser, B. Naturalist inquiry and grounded theory. *Forum: Qualitative Social Research 5*, 1 (2004).

[62] Glaser, B. Remodeling grounded theory. *Forum: Qualitative Social Research 5*, 2 (2004).

[63] Glaser, B. *The Grounded Theory Perspective III: Theoretical Coding.* Sociology Press, Mill Valley, CA, 2005.

[64] Glaser, B. Grounded Theory Institute: Methodology of Barney G. Glaser, 2010.

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

[66] Grisham, P. S., and Perry, D. E. Customer relationships and extreme programming. In *HSSE '05: Proceedings of the 2005 workshop on Human and social factors of software engineering* (New York, NY, USA, 2005), ACM, pp. 1–6.

[67] Grossman, F., Bergin, J., Leip, D., Merritt, S., and Gotel, O. One XP experience: introducing Agile (XP) software development into a culture that is willing but not ready. In *CASCON '04: Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research* (2004), IBM Press, pp. 242–254.

[68] Hanssen, G. K., and Faegri, T. E. Agile customer engagement: a longitudinal qualitative case study. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering* (New York, NY, USA, 2006), ACM, pp. 164–173.

[69] Hastie, S. Organizing Self-Organizing Agile Teams. InfoQ Article, url = http://www.infoq.com/news/2010/04/organizing-selforganizing-teams, last accessed on 9th Nov 2010.

[70] Highsmith, J. *Adaptive software development: a collaborative approach to managing complex systems.* Dorset House Publishing, New York, 2000.

[71] Highsmith, J. *Agile Project Management: Creating Innovative Products.* Addison-Weasley, USA, 2004.

[72] Highsmith, J., and Fowler, M. The Agile Manifesto. *Software Development Magazine 9*, 8 (2001), 29–30.

[73] Hoda, R., Noble, J., and Marshall, S. Negotiating contracts for Agile projects: A practical perspective. In *International Conference on Agile Software Development (XP)* (Italy, 2009), Springer, pp. 186–191.

[74] Hoda, R., Noble, J., and Marshall, S. Agile undercover: When customers don't collaborate. In *International Conference on Agile Software Development (XP)* (Norway, 2010), pp. 73–87.

[75] Hoda, R., Noble, J., and Marshall, S. Agility in context. In *OOPSLA* (Reno/Nevada, USA, 2010), ACM, pp. 74–88.

[76] Hoda, R., Noble, J., and Marshall, S. Balancing acts: Walking the Agile tightrope. In *Co-operative and Human Aspects of Software Engineering workshop at ICSE2010* (South Africa, 2010), ACM, pp. 5–12.

[77] Hoda, R., Noble, J., and Marshall, S. How Much is Just Enough? Some Documentation Patterns on Agile Projects. In *EuroPLoP2010* (Germany, 2010), Hillside Group.

[78] Hoda, R., Noble, J., and Marshall, S. Organizing self-organizing teams. In *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering* (South Africa, 2010), ACM, pp. 285–294.

[79] Hoda, R., Noble, J., and Marshall, S. Using grounded theory to study the human aspects of software engineering. In *Human Aspects of Software Engineering (HAoSE) workshop at SPLASH2010* (Reno/Nevada, USA, 2010), ACM, pp. 5:1–5:2.

[80] Hoda, R., Noble, J., and Marshall, S. What language does Agile speak? In *International Conference on Agile Software Development (XP)* (Norway, 2010), pp. 387–388.

[81] Hoda, R., Noble, J., and Marshall, S. Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Software Engineering* (2011), In Press.

[82] Hoda, R., Noble, J., and Marshall, S. The impact of inadequate customer collaboration on self-organizing agile teams. *Information and Software Technology 53* (May 2011), 521–534.

[83] Hoda, R., Noble, J., and Marshall, S. Supporting self-organizing agile teams: What's senior management got to do with it? In *International Conference on Agile Software Development (XP)* (Spain, 2011), ACM, p. To Appear.

[84] Hoeppner, K. D. Agile undercover. World Wide Web electronic publication, url = http://virtualbreath.net/curious/2010/08/23/agile-undercover/, last accessed on 9th Nov 2010.

[85] Horvath, N. Uses Cases & Scrum. World Wide Web electronic publication, `http://www.femara.com.br/media/12131/usecasesscrum.pdf`, last accessed on 9th Nov 2010.

[86] Hut, J., and Molleman, E. Empowerment and team development. *Team Performance Management 4*, 2 (1998), 53–66.

[87] Judy, K. H., and Krumins-Beens, I. Great Scrums need great product owners: Unbounded collaboration and collective product ownership. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences* (Washington, DC, USA, 2008), IEEE Computer Society, p. 462.

[88] Kauffman, S. A. *The Origins of Order.* Oxford University Press, New York, 1993.

[89] Kearns, A. Converting Waterfall Requirements into Underground Agile Features. World Wide Web electronic publication, url = http://www.morphological.geek.nz/blogs/viewpost/Peruse+Muse+Infuse/Converting+Waterfall+Requirements+into+Underground+Agile+Features.aspx, last accessed on 9th Nov 2010.

[90] Korkala, M., Abrahamsson, P., and Kyllonen, P. A case study on the impact of customer communication on defects in Agile software development. In *In Agile 2006* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 76–88.

[91] Lambert, A. Fluid families: A theoretical model for determining family membership within blended and ex-blended families. In *93rd Annual Convention of the NCA* (Chicago, IL, 2007).

[92] Lansing, S. J. Complex adaptive systems. *Annual Review of Anthropology 32* (2003), 183–204.

[93] Larman, C. *Agile and Iterative Development: A Manager's Guide.* Addison Wesley Professional, 2003.

[94] Larman, C., and Basili, V. R. Iterative and incremental development: A brief history. *Computer 36*, 6 (2003), 47–56.

[95] Larsen, D. Team Agility: Exploring Self-Organizing Software Development Teams. Industrial Logic and The Agile Times newsletter, `http://www.futureworksconsulting.com/resources/TeamAgilityAgileTimesFeb04.pdf`, last accessed on 14th Nov 2010.

[96] Levin, S. A. Ecosystems and the biosphere as complex adaptive systems. *Ecosystems 1*, 5 (1998), 431–436.

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

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

[99] Lewin, R. From chaos to complexity: Implications for organizations. *Executive Development 7*, 4 (1994), 16–17.

[100] Lindvall, M., Basili, V. R., Boehm, B. W., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L. A., and Zelkowitz, M. V. Empirical Findings in Agile Methods. In *In XP/Agile Universe* (London, UK, 2002), Springer-Verlag, pp. 197–207.

[101] Lowery, M., and Evans, M. Scaling product ownership. In *In Proceedings of the Agile 2007* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 328–333.

[102] Mamoli, S. Agile Undercover: When Customers don't Collaborate. World Wide Web electronic publication, url = http://www.nomad8.com/files/category-agile-product-ownership.php, last accessed on 9th Nov 2010.

[103] Manglaraj, G., Mahapatra, R., and Nerur, S. Acceptance of software process innovations the case of extreme programming. *European Journal of Information Systems 18*, 4 (2009), 344–354.

[104] Mann, C., and Maurer, F. A case study on the impact of Scrum on overtime and customer satisfaction. In *Agile Development Conference* (2005), IEEE Computer Society, pp. 70–79.

[105] Martin, A. *The role of the customers in Extreme Programming projects*. PhD thesis, School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, Wellington, New Zealand, 2009.

[106] Martin, A., Biddle, R., and Noble, J. The XP customer role in practice: Three studies. In *Agile Development Conference* (2004), pp. 42–54.

[107] Martin, A., Biddle, R., and Noble, J. The XP customer role: A grounded theory. In *In Agile 2009* (Chicago, 2009), IEEE Computer Society.

[108] Martin, R. *Agile Software Development: principles, patterns, and practices.* Pearson Education, NJ, 2002.

[109] Mills, H. D. *Software Productivity.* Little Brown and Company, 1983.

[110] Misra, S. C., Kumar, V., and Kumar, U. Identifying some important success factors in adopting Agile software development practices. *Journal of Systems Software. 82*, 11 (2009), 1869–1890.

[111] Moe, N. B., and Dingsoyr, T. Scrum and team effectiveness: Theory and practice. In *International Conference on Agile Software Development (XP)* (Limerick, 2008), Springer, pp. 11–20.

[112] Moe, N. B., Dingsøyr, T., and Dybå, T. A teamwork model for understanding an Agile team: A case study of a Scrum project. *Information and Software Technology 52*, 5 (2010), 480–491.

[113] Moe, N. B., Dingsoyr, T., and Dybå, T. Understanding self-organizing teams in Agile software development. In *ASWEC 08* (Washington, 2008), IEEE, pp. 76–85.

[114] Molleman, E. Variety and the requisite of self-organization. *International Journal of Organizational Analysis 6*, 2 (1998), 109–131.

[115] Morgan, G. *Images of organization.* Sage Publications, Beverly Hills, 1986.

[116] Myers, M. D. Qualitative research in information systems. *MIS Quaterly 21*, 2 (1997), 241–242.

[117] Nathaniel, K. A. *A Grounded Theory Of Moral Reckoning In Nursing.* PhD thesis, West Virginia University, 2003.

[118] Nerur, S., and Balijepally, V. Theoretical reflections on Agile development methodologies. *Commun. ACM 50*, 3 (2007), 79–83.

[119] Nerur, S. e. a. Challenges of migrating to Agile methodologies. *Commun. ACM 48*, 5 (2005), 72–78.

[120] Nonaka, I. A Dynamic Theory of Organizational Knowledge Creation. *Organization Science 5*, 1 (1994), 14–37.

[121] NVivo. Research software tool. World Wide Web electronic publication, `http://www.qsrinternational.com/products_nvivo.aspx`, last accessed on 10th April 2010.

[122] Oates, B. J. *Researching Information Systems and Computing.* Sage Publications, 2006.

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

[124] Palmer, S., and Felsing, M. *A Practical Guide to Feature- Driven Development.* Pearson Education, 2001.

[125] Parnell-Klabo, E. Introducing Lean principles with Agile practices at a Fortune 500 company. In *In Agile 2006* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 232–242.

[126] Parry, K. Grounded theory and social process: A new direction for leadership research. *Leadership Quaterly 9*, 1 (1998), 85–105.

[127] Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J. The impact of Agile practices on communication in software development. *Empirical Software Engineering 13*, 3 (2008), 303–337.

[128] Rising, L., and Janoff, N. S. The Scrum software development process for small teams. *IEEE Softw. 17*, 4 (2000), 26–32.

[129] Robey, D., Welke, R., and Turk, D. Traditional, iterative, and component-based development: A social analysis of software development paradigms. *Information Technology and Management 2*, 1 (2001), 53–70.

[130] Robinson, H., and Sharp, H. Organisational culture and XP: three case studies. In *Agile Development Conference* (2005), IEEE Computer Society, pp. 49–58.

[131] Robinson, H., and Sharp, H. The social side of technical practices. In *XP* (2005), pp. 100–108.

[132] Royce, W. W. Managing the development of large software systems: Concepts and techniques. In *ICSE* (1987), pp. 328–339.

[133] Ruparelia, N. B. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes 35*, 3 (2010), 8–13.

[134] Sawyer, S., Guinan, P. J., and Cooprider, J. Social interactions of information systems development teams: a performance perspective. *Information Systems Journal 20* (January 2010).

[135] Schein, E. H. *Organizational Culture and Leadership*, 1st edition ed. Jossey-Bass Publishers, San Franciso, 1985.

[136] Schwaber, K. Agile Processes and Self-Organization. Control Chaos, `http://www.controlchaos.com/download/Self%20Organization.pdf`, last accessed on 31st March 2010.

[137] Schwaber, K. Scrum Guide. Scrum Alliance Resources, `http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf`, last accessed on 9th Nov 2010.

[138] Schwaber, K., and Beedle, M. *Agile Software Development with SCRUM.* Prentice-Hall, 2002.

[139] Scrum Alliance. World Wide Web electronic publication, `http://www.scrumalliance.org/view/scrum_framework`, last accessed on Sep 16th, 2008.

[140] Sfetsos, P., Angelis, L., and Stamelos, I. Investigating the extreme programming system—an empirical study. *Empirical Software Engineering 11*, 2 (2006), 269–301.

[141] Sharp, H., and Robinson, H. An ethnographic study of XP practice. *Empirical Software Engineering 9*, 4 (2004), 353–375.

[142] Sharp, H., and Robinson, H. Collaboration and co-ordination in mature extreme programming teams. *International Journal of Human-Computer Studies 66*, 7 (2008), 506–518.

[143] Sjoberg, D. I., Dyba, T., and Jorgensen, M. The future of empirical methods in software engineering research. In *Future of Software Engineering* (2007), IEEE Computer Society.

[144] Sliger, M., and Broderick, S. *The Software Project Manager's Bridge to Agility.* Addison Wesley Professional, 2008.

[145] Software, M. G. Learning Scrum - free to use figures and wallpapers about Scrum. Online; last accessed 15-Nov-2010.

[146] Stapleton, J. *Dynamic Systems Development Method.* Addison Wesley, 1997.

[147] Stern, P. N. Eroding grounded theory. *Critical Issues in Qualitative Research Methods* (1994), 210–223.

[148] Strode, D. E., Huff, S. L., and Tretiakov, A. The Impact of Organizational Culture on Agile Method Use. In *Proceedings of the 42nd Hawaii International Conference on System Sciences* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 1–9.

[149] SUDDABY, R. From the editors: What grounded theory is not. *Academy of Management Journal 49*, 4 (2006), 633–642.

[150] SUMMERS, M. Insights into an Agile adventure with offshore partners. In *In Agile 2006* (USA, 2008), IEEE, pp. 333–338.

[151] SURESHCHANDRA, K., AND SHRINIVASAVADHANI, J. Adopting Agile in Distributed Development. In *ICGSE '08: Proceedings of the 2008 IEEE International Conference on Global Software Engineering* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 217–221.

[152] SUTHERLAND, J. Roots of Scrum: Takeuchi and self-organizing teams. World Wide Web electronic publication, `http://jeffsutherland.com`, last accessed on 31st March 2010.

[153] SUTHERLAND, J., SCHOONHEIM, G., RUSTENBURG, E., AND RIJK, M. Fully distributed Scrum: The secret sauce for hyperproductive offshored development teams. In *In Agile 2008* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 339–344.

[154] TAKEUCHI, H., AND NONAKA, I. The new new product development game. *Hardvard Business Review 64*, 1 (1986), 137–146.

[155] TAYLOR, P. S., GREER, D., SAGE, P., COLEMAN, G., MCDAID, K., AND KEENAN, F. Do Agile GSD experience reports help the practitioner? In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner* (New York, NY, USA, 2006), ACM, pp. 87–93.

[156] THOMAS, G., AND JAMES, D. Reinventing grounded theory: some questions about theory, ground and discovery. *British Educational Research Journal 32*, 6 (2006), 767–795.

[157] TOLFO, C., AND WAZLAWICK, R. S. The influence of organizational culture on the adoption of extreme programming. *Journal of Systems and Software 81*, 11 (2008), 1955–1967.

[158] TOMAYKO, J. E., AND HAZZAN, O. *Human Aspects of Software Engineering.* Charles River Media, Massachusetts, USA, 2004.

[159] TRIST, E. The evolution of socio-technical systems. *Occasional paper* (1981).

[160] TUCKMAN, B. W. Development Sequence in Small Groups. *Psychological Bulletin* (1965), 384–399.

[161] UY, E., AND IOANNOU, N. Growing and sustaining an offshore Scrum engagement. In *In Agile 2008* (USA, 2008), IEEE.

[162] WHITWORTH, E., AND BIDDLE, R. The social nature of Agile teams. In *In Agile 2007* (USA, 2007), IEEE Computer Society, pp. 26–36.

[163] WIKIPEDIA. Spiral model, 2010. [Online; last accessed 15-Nov-2010].

[164] WILLIAMS, L., KESSLER, R. R., CUNNINGHAM, W., AND JEFFRIES, R. Strengthening the case for pair programming. *IEEE Softw. 17*, 4 (2000), 19–25.

[165] WILLIAMS, L. A., AND KESSLER, R. R. All I really need to know about pair programming I learned in kindergarten. *Commun. ACM 43*, 5 (2000), 108–114.

[166] XP. Extreme programming: A gentle introduction. World Wide Web electronic publication, `http://www.extremeprogramming.org/`, last accessed on 23rd Oct 2010.

[167] YIN, R. *Case Study Research: Design and Methods.* Sage Publications, CA, 1984.