Balancing Acts: Walking the Agile Tightrope

Rashina Hoda
School of Engineering and Computer Science
Victoria University of Wellington
New Zealand
rashina@ecs.vuw.ac.nz

James Noble
School of Engineering and Computer Science
Victoria University of Wellington
New Zealand
kjx@ecs.vuw.ac.nz

Stuart Marshall
School of Engineering and Computer Science
Victoria University of Wellington
New Zealand
stuart@ecs.vuw.ac.nz

ABSTRACT

Self-organizing teams are one of the critical success factors on Agile projects - and yet, little is known about the self-organizing nature of Agile teams and the challenges they face in industrial practice. Based on a Grounded Theory study of 40 Agile practitioners across 16 software development organizations in New Zealand and India, we describe how self-organizing Agile teams perform balancing acts between (a) freedom and responsibility (b) cross-functionality and specialization, and (c) continuous learning and iteration pressure, in an effort to maintain their self-organizing nature. We discuss the relationship between these three balancing acts and the fundamental conditions of self-organizing teams - autonomy, cross-fertilization, and self-transcendence.

General Terms

Software Engineering, Human Factors, Theory

Keywords

Software Engineering, Self-organizing teams, Agile software development

1. INTRODUCTION

With the increasing popularity of Agile software development in software engineering [7, 24], several researches have explored various aspects of Agile teams [10, 22, 24, 28]. The self-organizing nature of Agile teams, however, remains largely ignored with no substantial study on the subject across multiple projects, organizations, and cultures. Our contribution in this paper is to understand the self-organizing nature of Agile teams and the balancing acts they perform in an effort to maintain their self-organizing nature.

Agile software development methods follow an iterative and incremental style of development where collaborative self-organizing teams dynamically adjust to changing customer requirements [1, 12, 19, 21]. The Agile Manifesto [18] defines the four basic values of Agile methods as:

"*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."

Self-organizing teams are one of the twelve principles behind the Manifesto [18]. Scrum [27] and XP (eXtreme Programming) [6] are the most widely adopted Agile methods in the world [25]. Scrum mainly covers project management [12] and XP focuses on developmental practices.

Agile teams are self-organizing teams [9, 11, 18, 21, 26]. Self-organizing teams manage their own work and organize around the details of their tasks. Self-organizing 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.*" [17]. Self-organizing teams must have common focus, mutual trust, respect, and the ability to organize repeatedly to meet new challenges [11].

Self-organizing teams have been identified as one of the critical success factors of Agile projects [9]. Self-organization can also directly influence team effectiveness [22] as self-management brings decision making authority to the level of operational problems, which increases the speed and accuracy of problem solving. Sharp and Robinson [29] note that mature XP teams are highly collaborative and self-organizing in nature. Takeuchi and Nonaka [31] describe self-organizing teams as exhibiting autonomy, cross-fertilization, and self-transcendence. Moe et al. note that Scrum emphasizes self-organizing teams but does not provide clear guidelines on how they should be implemented [22].

Self-organizing teams are not leaderless, uncontrolled teams [11, 31]. Leadership in self-organizing teams is meant to be *light-touch* and *adaptive* [5], providing feedback and subtle direction [3, 8, 31]. Leaders of Agile teams are responsible for setting direction, aligning people, obtaining resources, and motivating the teams [3]. Agile projects have job titles such as Scrum Masters [27] and (XP) Coaches [13] instead of traditional managers.

2. RESEARCH CONTEXT AND METHOD

We used Grounded Theory [15] as our research method as it enables the study of social interactions and behaviour — a focus of Agile software development. We interviewed 40 Agile practitioners across 16 different software organizations in New Zealand and India over a period of 2.5 years.

Data was collected through face-to-face, semi-structured interviews using open-ended questions. The interviews were approximately an hour long and focused on the challenges faced on Ag-

Table 1: Participants and Projects (Agile Positions: Agile Coach (AC), Developer (Dev), Customer Rep (Cust Rep), Business Analyst (BA), Senior Management (SM); Organizational Size: XS < 50, S < 500, M < 5000, L < 50,000, XL > 100,000 employees; Project duration is in months & Iterations are in weeks.)

Participants	Agile Position	Agile Method	Org. Size	Country	Domain	Team Size	Project Duration	Iteration
P1-P7	Dev × 3, BA, AC, Tester, Cust Rep	Scrum	M	NZ	Health	7	9	2
P8	AC	Scrum & XP	L	NZ	Social Services	4 to 10	3 to 12	2
P9-P15	Dev × 5, AC, SM	Scrum & XP	S	NZ	Environment	4 to 6	12	1
P16	SM	Scrum & XP	S	NZ	E-commerce	4	2	4
P17	AC	Scrum & XP	XL	NZ	Telecom & Transportation	6 to 15	12	4
P18	Cust Rep	Scrum	XS	NZ	Entertainment	6 to 8	9	4
P19	AC	Scrum & XP	S	NZ	Government Education	4 to 9	4	2
P20	Dev	Scrum & XP	XS	NZ	Software Development	7	6	2
P21-P27	Dev × 4, AC, Tester, SM	Scrum & XP	S	India	Software Development & Consultancy	5	6	2
P28-P31	Dev × 4	Scrum & XP	XS	India	Software Development	4	1	1
P32	AC	Scrum & XP	S	India	IT & Agile Training	7 to 8	48	3
P33-P36	AC × 4	Scrum & XP	M	India	Software Development	7 to 8	3 to 6	2
P37	AC	Scrum & XP	M	India	Financial Services	8 to 11	36	2
P38	Designer	Scrum & XP	S	India	Web-based services	5	1	2
P39	AC	Scrum & XP	L	India	Telecom	8 to 15	3	4
P40	SM	Scrum & XP	M	India	Software Development	15	12	1

ile projects and the strategies used to overcome them. In order to get a rounded perspective of Agile projects, we interviewed and observed project participants employed as part of the Agile team (Agile Coaches including Scrum Masters and XP Coaches, Developers, Designers, Testers, and Business Analysts) or as Customer Representatives (Product Owners), and Senior Management. All the teams we studied used Agile development methods, primarily Scrum and XP. The teams used Agile practices such as iterative development, release and iteration planning, test-driven development, daily stand-up meetings, frequent delivery of software, and continuous integration. Table 1 shows participant and project details.

The project duration varied from 2 to 12 months and the team sizes varied from 2 to 20 people on different projects. The products and services offered by the participants' organizations included web-based applications, front and back-end functionality, and local and off-shored software development services. Half the participants were practicing in India and half in New Zealand. The organizational sizes varied from 10 to 300,000 employees. In order to respect their confidentiality, we refer to our participants by numbers P1 to P40.

The data collected through interviews was strengthened by our observation of several Agile practices on at least two projects in New Zealand and a couple in India. We attended and observed Agile practices such as daily stand-up meetings (co-located and distributed), release planning, iteration planning, and demonstrations. We conducted data collection and analysis iteratively so that constant comparison of data helped guide future interviews and the analysis of interviews and observations fed back into the emerging results.

Open coding [16] was used to analyze the interview transcripts in detail. Using GT's *constant comparison method* data from interview transcripts was raised in levels of abstraction from key points to codes to concepts to categories [15, 16]. Details of our data analysis can be found at [20] and are not provided here due to space reasons. Our study focuses on self-organization in Agile teams. As with all GT studies, we do not claim our results to be universally applicable, rather they accurately characterize the context studied.

The term *Balancing Acts* emerged as a result of our data analysis to describe the efforts made by self-organizing teams to balance between different (and often contrasting) concepts, such as between freedom and responsibility. In the following sections, we describe these balancing acts. In each of the balancing acts we describe how Agile teams were able to achieve a balance and the consequence of imbalance. We have selected quotations drawn from our interviews that serve to highlight the results and that are spread across most participants.

3. BALANCING FREEDOM & RESPONSIBILITY

Participants of our research study mentioned that they experienced more freedom as a part of a self-organizing Agile team than they had as a part of a traditional software development team. They recalled that their previous traditional software development projects involved managers setting team goals, assigning individual tasks for the team members to achieve within set time-frames, and micromanaging the projects on a day to day basis. They found such an environment to be frustrating and demotivating.

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

Agile teams need senior management support, in terms of freedom, to manage their own affairs. Agile teams facing a management that was still dictating terms are 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.*" - P25, Agile Coach, India

Agile teams were provided freedom by their senior management to organize themselves, giving them a concrete sense of empowerment because of their ability to self-assign, self-commit, self-manage, self-evaluate, and self-improve. Participants found that the "*team is making a lot more decisions*" (P8) and "*every person is contributing to the decision-making*" (P20).

"*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.*" — P25, Agile Coach, India

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.*" (P34). Participants mentioned that team members were "*putting their hand up to do stuff*", they "*get better [at] organization*", and at the same time there was "*a lot more ownership*" and "*sense of responsibility and accountability*" (P20, P32). Agile teams are aware of their responsibility to adhere to Agile practices, responsibility towards other each other, and responsibility to achieve team goals. There were many project-related activities that required self-organizing teams to perform the balancing act between freedom and responsibility. We describe two most common ones below:

When Committing to a Team Goal

Agile methods require customer representatives to provide project requirements in the form of user stories. These user stories are broken down into developmental level tasks by the teams [6, 27]. While Agile methods grant customers the ability to prioritize user stories every iteration, the decision of how many stories will be attempted in an iteration (developmental pace or team velocity) is ideally the team's decision, based on their capability:

"*We have stories which we estimate complexity of and we say 'well, we can fit this much complexity into next two weeks'*" — P8, Agile Coach, NZ

We found that Indian teams enjoyed the same freedom to plan their iterations and commit to a team goal as their NZ counterparts:

"*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.*" — P26, Tester, India

We found that teams in both cultures enjoyed the freedom to set the team goal and at the same time realized their responsibilities to ensure they achieved the set iteration goal as a team through a collaborative effort:

"*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, NZ

"*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.*" — P28, Developer, India

During Self-Assignment

Committing to a team goal every iteration is the group decision. Self-assignment of tasks within the committed user stories, on the other hand, is an individual decision. Most participants appreciated the freedom they had to be able to pull the tasks from the story board and self-assign them (P9, P11, P21, P24, P25, P28, P30, P32).

"*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.*" — P21, Developer, India

Teams have the freedom to pick tasks but they also bear the responsibility to pick tasks based on business value first, as recommended by Agile methods:

"*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.*" — P24, Developer, India

In situations were several tasks are of the same business value or priority, individuals realize their responsibilities towards other team members and avoid picking tasks based on ease of implementation. The story board - a physical paper chart posted on the wall; containing all stories, tasks, their states, and names of individuals working on them - provides high visibility and transparency about task assignments [29] such that "*when someone looks to the board they can see who is working on which task*" (P9). The visibility and 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*" — P32, Agile Coach, India

Individuals try to avoid potential conflicts around task dependencies. For instance, members in some teams unofficially announced the task as they picked it from the board such that any potential conflict could be easily raised by others and mutually resolved. Such actions displayed responsibility towards team members.

We found some Indian teams struggled to take ownership of tasks during their early stages of Agile adoption:

"*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*" — P24, Developer, India

Initially, we suspected this to be the effect of the Indian hierarchical culture [4, 30, 32] with a low individualism (IDV) score and high Power Distance Index (PDI) [2] where managers are expected to make all decisions. Our analysis of NZ teams, however, disclosed that NZ teams new to Agile methods faced the same challenge:

"*It took them [new Agile team] a bit of time to stop coming and asking us what they should be working on and the answer was always 'pick one!' And after while it became natural...people were picking stuff...and that worked really well.*" — P20, Senior Developer, NZ

It emerged that the initial struggle to accept freedom and use it with responsibility was not based on cultural differences, rather it was 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 proactive and do things for themselves" (P13) and "*assign[ing] to themselves needs maturity*" (P33). Relatively inexperienced Agile teams had issues with accepting autonomy and kept looking up to their seniors and Agile coaches for guidance and decision making. More mature Agile teams (fluent in use of Agile practices, for usually more than a year), however, valued their freedom and were able to effectively balance the freedom they had been provided by senior management with the responsibility.

Consequence of Imbalance

The importance of balancing freedom and responsibility was most apparent when a team was unable to use their freedom in a responsible manner forcing senior management to intervene. For example, the general manager of an Agile organization in India shared an experience where they had to interfere with a self-organizing team which was unable to balance between freedom and responsibility. The team had a couple of senior developers who were extremely proficient at their tasks but were misusing the freedom provided by the senior management to dictate and override the rest of the team. Their influence had become so strong that it led to a clear unofficial divide in the team between those that sided with them in every decision fearing fallback 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* [20], sought senior management interference and removed those senior developers from the team. It took the rest of the team some time to return to their previous self-organizing nature. The consequence of imbalance between freedom and responsibility is senior management intervention, which may lead to restrictions on the team's self-organization.

4. BALANCING CROSS-FUNCTIONALITY & SPECIALIZATION

We found that Agile teams make an effort to instill and maintain cross-functionality in the team. Cross-functionality is the ability of team members to look beyond their area of specialization by taking an interest in activities outside their specialization. Cross-functionality allows team members to gain a more rounded vision of the project through understanding it from multiple perspectives.

We found that while Agile teams generally promote cross functionality, they cannot completely dispose off specialization, as we describe later in this section. Therefore, most teams tried to balance between cross-functionality and specialization across (a) different functional roles, such as between developers and testers, and across (b) different technical areas of expertise, such as between database management and graphical user interface design. We describe these in the following subsections.

Across Functional Roles

Cross-functionality across functional roles was common among the participants and was considered to be one of the key characteristics of Agile teams:

> "*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.*" — P17, Agile Coach, NZ

Team members in different functional 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. In particular, we found developers and testers were not pitched against each other, rather they interacted frequently in the interest of the project, leading to cross-functionality:

> "*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

Understanding each others' perspectives meant team members could help with some parts of other functional areas within the limitations of their cross-functional abilities and after they had tended to their own specialized tasks:

> "*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 progressed 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, NZ

> "*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.* – P25, Agile Coach, India

Across Technical Areas of Expertise

Cross-functionality across different areas of expertise (within the same functional role) allowed team members to become familiar with most technical aspects of the project so they could easily manage any area, which is consistent with XP's collective code ownership principle [6].

> "*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

Flexibility to work in multiple technical areas was welcomed by developers because it helped them maintain interest in their work. As one of the developers nearing an end to 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

Developers in India were equally excited by the ability to pick tasks across different technical areas:

> "*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.*" — P22, Developer, India

Venturing outside the areas of technical expertise was not always easy. An Agile Coach in India noted that his team members were uncomfortable in practicing cross-fertilization because "*now they are switching role...people don't want to come into different shoes, different hats very frequently*" (P36). The reason behind this hesitance may be, as a developer in New Zealand mentioned, the fear of exposing inadequacies:

> "*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.*" — P9, Developer, NZ

The culture of collaboration and cooperation in an Agile team helps team members overcome such apprehensions and explore other areas of expertise. Most Agile teams we studied were 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, NZ

Consequence of Imbalance

The example below serves to highlight the need for balancing between specialization and cross-functionality when working in a self-organizing team. A business analyst in a NZ team secretly misused their cross-functional programming skills and had started causing damage to the project code base. The BA's unofficial involvement in programming had caused the team several hours of rework. The Agile Coach took on a *Terminator* role [20] - 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.*" — P7, Customer Representative, NZ

A team's failure to balance between cross-functionality and specialization, as in this example, can invite senior management intervention. Frequent senior management interference can pose a threat to the team's self-organizing nature.

5. BALANCING CONTINUOUS LEARNING & ITERATION PRESSURE

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.*" — P20, Senior Developer, NZ

> "*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, NZ

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.*" — P10, Developer, NZ

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.*" — P6, Tester, NZ

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, NZ

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. We now describe how Agile teams attempt to achieve the difficult task of balancing continuous learning of different types alongside the pressure to deliver the team goal every iteration.

Through Pair-in-Need

Pair programming, a particular form of collaboration, is a standard XP practice where developers work in pairs on every task [6]. Our participants, practicing combinations of Scrum and XP, used a practice we call *Pair-in-Need* — where pairing was done on a need basis to "*distribute knowledge*" (P21) and complete complex tasks.

> "*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.*" — P25, 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 [20] to help new comers learn the basic Agile practices and catch up to the team's velocity.

> "*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 [to catch up to team velocity]. But [my team] held 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.*" — P26, Tester, NZ

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 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 worked well for these teams because it allowed them to both learn how to tackle new and complex tasks with the help of a peer and at the same time move closer to their set iteration goal.

Through Retrospectives

Retrospectives are often used an 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, NZ

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.*" — P17, Agile Coach, NZ

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*" (P6). Participants found retrospectives to be "*a key ingredient in Agile methodology*" (P26) which allowed them to evaluate team practices, including team velocity, and correcting them as needed.

Consequence of Imbalance

A developer shared an experience where their team had committed to too much in an iteration 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!*" – P20, Senior 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:

> "*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!*" – P20, Senior Developer, NZ

The team also decided to learn and setup better guidelines for testing. Another 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 to succumb to iteration pressure and the team took the time to simply work on needed improvement in 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.*" — P14, Agile Coach, NZ

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

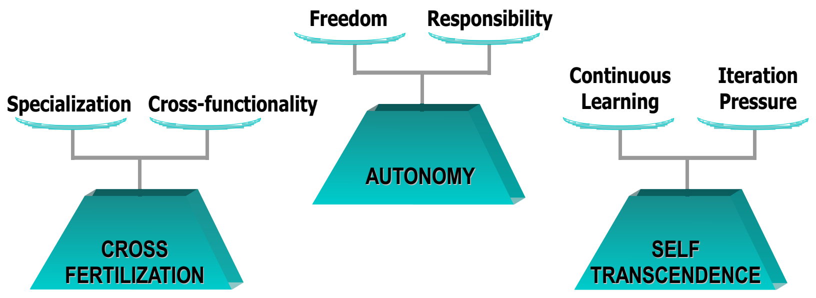

Figure 1: Relationship between the Balancing Acts and the fundamental conditions of self-organization.

6. DISCUSSION

Following Grounded Theory, we first collected and analyzed the data. Once the findings seemed sufficiently grounded and developed, we then reviewed the literature on self-organizing Agile teams. 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 [14].

We found that the subject of self-organizing teams has largely been ignored in Agile research literature. One of the earliest papers to mention and describe self-organizing teams was "the new new product development game" by Nonaka et al. [31], where they define a group to possess self-organizing capability when it exhibits 3 conditions: autonomy, cross-fertilization, and self-transcendence. After careful study of the three conditions of self-organizing teams, we found relationships between those conditions and the balancing acts found as a result of our study. We discovered that 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. Figure 2 depicts the relationships between the balancing acts and the fundamental conditions of self-organizing teams as found in literature [31]. We discuss each of these relationships below.

Autonomy

According to Nonaka et al., 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 [31]. We found that participants were provided freedom by senior management to manage their own tasks which fulfills the first criteria of autonomy. In order to ensure there was minimum interference from senior management, the second criteria 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

Nonaka et al. found that 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 other's perspectives [31]. In our study, we found that teams consisted of individual members with varying specializations — developers, testers, business analysts — which fulfills the first criteria for cross-fertilization. In order to ensure that these individuals benefited from understanding each others' perspectives, the second criteria of cross-fertilization, the teams frequently interacted across different functional roles and attempted tasks across different technical areas. Teams found it impossible to completely avoid specialization but tried to be as cross-functional as possible. The teams' ability to balance specialization and cross-functionality meant they could achieve and sustain cross-fertilization.

Self-Transcendence

According to Nonaka et al., 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. In our study, we found that teams were able to establish their own goals in terms of deciding how much to commit to in an iteration, thus fulfilling the first criteria of self-transcendence. Teams not only established their own goals but also assumed full responsibility to achieve those goals causing pressure to deliver. While some iteration pressure motivated teams to achieve their goals, excessive pressure resulted in a neglect of learning and improvement. In order to balance between iteration pressure and the need for continuous learning, the teams practiced *Pair-in-need* to both complete tasks and learn from each other in the process. The other technique was to engage in retrospective meetings to self-evaluate and suggest ways of improvement. Teams used retrospectives to find a balance between the amount of time they devoted to finishing tasks versus the time they would spend specifically on learning new and better ways of working. Thus, by balancing between iteration pressure and the need for continuous learning, teams were able to achieve self-transcendence.

7. RELATED WORK

Research on self-organizing nature of Agile teams is scarce. In a single case study, Moe et al. studied barriers to self-organization by focusing on one aspect of self-organization — autonomy [23]. They found that the management did not provide an environment conducive to self-organization that led to reduced external autonomy. Their findings reflect our assertion that senior management support, in terms of providing freedom, is important for teams to self-organize. They claim 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 teams' individualistic culture [4] did not negatively affect collaboration and co-ordination on these teams. The team's inability to balance freedom and whole team responsibility, however, can be an important barrier to self-organization.

8. LIMITATIONS

Since the codes, concepts, and category emerged directly from the data, which in turn was collected directly from real world, the results are grounded in the context of the data [20]. We do not claim the results to be universally applicable: rather, they accurately characterize the context studied [20]. Our choice of research destinations and participants were limited in some ways by our access to them.

9. CONCLUSION

We conducted a Grounded Theory study of 40 Agile practitioners across 16 different software development organizations in New Zealand and India. We found that Agile teams perform balancing acts between (a) freedom provided by senior management and responsibility expected from them in return; (b) specialization and cross-functionality across different functional roles and areas of technical expertise; and (c) continuous learning and iteration pressure. These balancing acts were performed by Agile teams in an effort to uphold the fundamental conditions of self-organizing teams — autonomy, cross-fertilization, and self-transcendence, respectively. These three balancing acts were not easy to perform but, when done well, ensured the teams were able to sustain their self-organizing nature. We hope our findings will help Agile teams and their management better comprehend ways to achieve and sustain the fundamental conditions of self-organizing teams through these balancing acts.

10. ACKOWLEDGMENTS

We thank all our participants. This research is supported by an Agile Alliance academic grant and a NZ BuildIT PhD scholarship.

11. REFERENCES

[1] N. Abbas et al. Historical roots of agile methods: Where did “agile thinking” come from? In *XP*, 94–103, Springer, Limerick, 2008.

[2] L. R. Abraham. Cultural differences in software engineering. In *ISEC ’09:*, 95–100, ACM, New York, 2009.

[3] L. Anderson et al. Agile management - an oxymoron?: who needs managers anyway? In *OOPSLA Comp.:*, 275–277, ACM, 2003.

[4] J. Aston, L. Laroche, and G. Meszaros. Cowboys and Indians: Impacts of cultural diversity on agile teams. In *AGILE ’08:* , 423–428, IEEE, Washington, 2008.

[5] S. Augustine, B. Payne, F. Sencindiver, and S. Woodcock. Agile project management: steering from the edges. *Commun. ACM*, 48(12):85–89, 2005.

[6] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, USA, 1999.

[7] A. Begel and N. Nagappan. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *ESEM ‘07:* , 255–264, IEEE, Washington, 2007.

[8] T. Chau and F. Maurer. Knowledge sharing in agile software teams. *Logic versus approximation:* LNCS 3075:173–183, 2004.

[9] T. Chow and D. Cao. A survey study of critical success factors in agile software projects. *J. Syst. Softw.*, 81(6):961–971, 2008.

[10] A. Cockburn. *People and Methodologies in Software Development*. PhD thesis, University of Oslo, Norway, 2003.

[11] A. Cockburn and J. Highsmith. Agile software development: The people factor. *Computer*, 34(11):131–133, 2001.

[12] T. Dybå and T. Dingsoyr. Empirical studies of Agile software development: A systematic review. *Inf. Softw. Technol.*, 50(9-10):833–859, 2008.

[13] S. Fraser et al. Xtreme programming and Agile coaching. In *OOPSLA Comp. ’03:*, 265–267, ACM, New York, 2003.

[14] B. Glaser. Theoretical Sensitivity. Sociology Press, Mill Valley, California, 1978.

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

[16] S. Georgieva and G. Allan Best Practices in Project Management Through a Grounded Theory Lens. *Electronic Journal of Business Research Methods*, 2008.

[17] J. Highsmith. *Agile Project Management: Creating Innovative Products*. Addison Wesley, USA, 2004.

[18] J. Highsmith and M. Fowler. The Agile Manifesto. *Software Development Magazine*, 9(8):29–30, 2001.

[19] R. Hoda, J. Noble, and S. Marshall. Negotiating contracts for Agile Projects: A Practical Perspective In *XP2009:* 186-191, Italy, 2009.

[20] R. Hoda, J. Noble, and S. Marshall. Organizing Self-Organizing Teams (To appear) In *ICSE2010:* Cape Town, 2010.

[21] R. Martin. *Agile Software Development: principles, patterns, and practices.* Pearson Education, NJ, 2002.

[22] N. B. Moe and T. Dingsoyr. Scrum and team effectiveness: Theory and practice. In *XP*,(9) 11–20, Limerick, Springer, 2008.

[23] N. B. Moe, T. Dingsoyr, and T. Dybå. Understanding self-organizing teams in agile software development. In *ASWEC ’08:*, 76–85, IEEE, Washington, 2008.

[24] S. Nerur et al. Challenges of migrating to agile methodologies. *Commun. ACM*, 48(5):72–78, 2005.

[25] M. Pikkarainen et al. The impact of agile practices on communication in software development. *Empirical Softw. Engg.*, 303-337, 2008.

[26] K. Schwaber *Scrum Guide*. Scrum Alliance Resources, 2009.

[27] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, NJ, USA, 2001.

[28] H. Sharp and H. Robinson. An ethnographic study of XP practice. *Empirical Softw. Engg.*, 9(4):353–375, 2004.

[29] H. Sharp and H. Robinson. Collaboration and co-ordination in mature extreme programming teams. *Int. J. Hum.-Comput. Stud.*, 66(7):506–518, 2008.

[30] M. Summers. Insights into an Agile adventure with offshore partners. In *AGILE ’08:*, 333–338, IEEE, USA, 2008.

[31] H. Takeuchi and I. Nonaka. The new new product development game. *Harvard Business Review*, 1986.

[32] E. Uy and N. Ioannou. Growing and sustaining an offshore Scrum engagement. In *AGILE ’08:*, IEEE, USA, 2008.