# Agility in Context

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

Philippe Kruchten
Electrical and Computer Engineering
University of British Columbia
Vancouver
Canada
pbk@ece.ubc.ca

James Noble
Stuart Marshall
Engineering and Computer Science
Victoria University of Wellington
Wellington
New Zealand
kjx,stuart@ecs.vuw.ac.nz

## Abstract

Evangelists for Agile methods strongly encourage all projects to follow every practice of their chosen method. Based on a Grounded Theory study involving 40 participants at 16 organizations, and corroborated by 4 independent case studies, we argue that development methods and practices must be adapted to fit their contexts. Understanding Agility in context will help development teams, their managers, and Agile coaches to adapt development processes to fit their projects' contexts.

***Categories and Subject Descriptors*** K.6.1 [*Project and People Management*]: Management techniques; K.6.3 [*Software Management*]: Software development/process

***General Terms*** software development, human factors, management

***Keywords*** Agility, Context, Agile Software Development, Adaptation

## 1. Introduction

Agile evangelists exhort teams to adopt their methods whole. For example, every project following Scrum must adopt every practice, enacted precisely as described in the Scrum manuals, books and courses [77]. Projects that do not follow these methods "by-the-book" are derided as "Scrum-butts", and invited to measure "*the Ten Ways You are Not AGILE*", or take the Nokia Test [18, 77, 79]. After all, if you are almost doing XP, then you are not doing XP [27]. Practitioners must have the courage to try [9] because "*Scrum works in any environment and can scale to programming in the large*" [78], and if pain persists then *"you first have to believe"* [48].

In contrast, an increasing number of practitioners and researchers support a contextual approach to Agile development, where Agile methods are adapted to suit their context [2, 24, 25]. Conboy and Fitzgerald study of experts' opinion on Agile methods notes that "*the very name agile suggests that the method should be easily adjusted to suit its environment*" [25].

We have conducted a large-scale Grounded Theory study of Agile practices involving 40 Agile practitioners from 16 software development organizations in New Zealand (NZ) and India [42–46]. In this paper, we present the results of our study as they relate to the adaptation of Agile methods to suit their projects' context. We have corroborated our study by four independent case studies.

Agile methods work well for projects within particular contexts: small; co-located teams; customers (product owners) who can make decisions on requirements; requirements that change over weeks or months; variable scope or variable price contracts; and few legal or regulatory constraints on development processes. Our study uncovered how development teams evolve Agile practices to fit other contexts — offshore or distributed development; projects without customers to provide requirements; requirements or systems that change rarely; or the legal necessity of certification of product and process. By fitting their practices to their project's context, teams can cleave to the *principles* of Agile development, even if they depart from one or more particular *practices*.

The rest of the paper is organized as follows: we briefly describe the principles of Agile software development in section 2, followed by a description of our research methodology and limitations in section 3. Section 4 presents our research findings, and then section 5 discusses the implications of our findings. Section 6 concludes the paper.


*OOPSLA/SPLASH'10,* October 17–21, 2010, Reno/Tahoe, Nevada, USA.

## 2. Background: Agility in Principle

Agile software development methods emerged in the late 1990s [52]. The term "Agile" was adopted as the umbrella term for methods such as Scrum [72], XP (eXtreme Programming) [9], Crystal [16], FDD (Feature Driven Development) [62], DSDM (Dynamic Software Development Method) [74], and Adaptive Software Development [40]. Agile methods are characterized by iterative and incremental development and promote frequent delivery of product features that are prioritized in consultation with the customers, aiming to deliver business value in each iteration. Agile methods address small, co-located, dedicated, and highly collaborative teams [12, 29, 61]. Scrum and XP are the most widely adopted Agile methods [63]: XP focuses on developmental practices, while Scrum mainly covers project management [29].

Agile methods are well-suited to projects with highly volatile requirements - they encourage projects to "*embrace change*" [9] and "*responding to change*" [41]. Adolph has defined Agility as "*the ability of an organization to react to change in its environment faster than the rate of these changes*" [3].

The developers of some of these Agile methods collaboratively wrote the Agile Manifesto:

> "*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 [41] include fast, frequent, consistent, and continuous delivery of working software; responding to changing requirements; encouraging effective (preferably face-to-face) communication; and motivated and well-supported self-organizing teams. The last principle specifically supports regular self-inspection and adaptation: "*At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly*" [41].

## 3. Research Method

### 3.1 Grounded Theory Study

Grounded Theory (GT) is the systematic generation of theory from data acquired by a rigorous qualitative research method [35, 36]. GT was developed by sociologists Glaser and Strauss [38]. GT does not involve hypothesis testing, rather the research product constitutes a theoretical formulation or integrated set of conceptual hypotheses about the substantive area under study [34].

We chose to use GT as our research method for several reasons [44]. Firstly, Agile methods focus on people and interactions, and GT, used as a qualitative research method, allows us to study social interactions and behaviour. Secondly, GT is suited to areas of research which have not been explored in great detail before, and the research literature on contextualization of Agile methods is scarce. Finally, GT is being increasingly being used to study Software Engineering [8, 13, 14, 50] and in particular, Agile methods [5, 20, 23, 44, 56, 83]. Using GT, we have applied a rigorous research method to study practical applications of Agile methods and to analyze and explain the results. We started out with a general area of interest — Agile project management — instead of beginning with a specific research question [23, 35].

***Data Collection*** We interviewed 40 Agile practitioners from 16 different software organizations in 2 countries — New Zealand and India. We interviewed participants from a range of different roles within Agile projects, so as to ensure that we had a rounded perspective of how their Agile teams worked. In particular, we interviewed and observed Agile Coaches (Scrum Masters and XP Coaches), Developers, Designers, Testers, Business Analysts, Product Owners, and Senior Management. All the teams we studied have adopted Agile development methods, primarily Scrum. 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. The first part of table 1 shows participant and project details for the Grounded Theory Research Study.

The projects' durations varied from 1 to 48 months and the team size varied from 4 to 15 people on different projects. The products and services that the participants' organizations offered included web-based applications, front and back-end functionality, and local and off-shored software development services. Half the participants were practising in India and half in New Zealand. The organizational sizes varied from 10 to 300,000 employees. To preserve confidentiality, we refer to our participants by numbers P1 to P40.

We conducted face-to-face, semi-structured interviews using open-ended questions. The interviews were approximately an hour long and were scheduled at the practitioners' workplaces or mutually agreed public locations. The interview questions focused on the participants' experiences of working with Agile methods and in particular, we asked about the challenges they faced in Agile projects and the strategies they used to overcome them. Although the answers varied with the individual participants, we later discovered during analysis that projects' contexts lead teams to adjust their practices.

The interview data was strengthened by our observation of several Agile practices on two projects in New Zealand and two in India. We attended and observed Agile practices such as daily stand-up meetings (co-located and distributed), release planning, iteration planning, and demonstrations of the teams from which our interview participants were de-

**Table 1.** Participants and Projects (P#: Participant Number, Agile Position: Agile Coach (AC), Agile Trainer (AT), 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) *a la carte [25]

Grounded Theory Research Study

| P# | Positions | Method | Org. Size* | Location | Domain | Team Size | Project (months) | Iteration (weeks) |
|---|---|---|---|---|---|---|---|---|
| P1-P7 | Dev X 3, BA, AC, Tester, Cust Rep | Scrum | M | NZ | Health | 7 | 9 | 2 |
| P8 | Cust Rep | Scrum & XP | L | NZ | Social Services | 4 to 10 | 3 1o 12 | 2 |
| P9-P15 | Dev X 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 | AC | Scrum & XP | XS | NZ | Software Development | 8 | 12 | 1 |
| P21-P27 | Dev X 4, AC, Tester, SM | Scrum & XP | S | India | Software Development & Consultancy | 5 | 6 | 2 |
| P28-P31 | Dev X 4 | Scrum & XP | XS | India | Software Development | 4 | 1 | 1 |
| P32 | AT | Scrum & XP | XS | India | Agile Training | 7 | 8 | 3 |
| P33-P36 | AC X 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 | Dev | Scrum & XP | M | India | Software Development | 15 | 12 | 1 |

Corroborating Case Studies

| Case Studies | Positions | Method | Org. Size* | Location | Domain | Team Size | Project (months) | Iteration (weeks) |
|---|---|---|---|---|---|---|---|---|
| FIN | SM, Architect | Scrum & XP | L | USA | Finance | 50 | 36 | 1 |
| FACT | Management & other roles | a la carte* | XL | Canada | Process Control | 6 to 15 | 12 | variable |
| FLY | All roles | Adapted RUP | XL | USA | Aeronautics | 6 to 25 | 12 | 4 |
| MATH | Project leads | XP | XS | Canada | Trade Analysis | 3 to 8 | 12 | 1 to 2 |

rived. In order to maintain consistency in the application of Grounded Theory, all data was collected and analyzed personally by the primary researcher (first author).

***Data Analysis*** We used open coding to analyze the interview transcripts in detail [7, 33]. In order to illustrate GT analysis, we present an example of working from interview transcripts to results for one of the categories, *Lack of Customer Involvement*. Other examples of our GT analysis are described elsewhere [43, 44, 47].

We began by collating key points from each interview transcript [33]. We then assigned a *code* to each key point. A *code* is a phrase that summaries the key point in 2 or 3 words.

**Interview quotation**: "*Big, big issues are getting enough collaboration time with the [customers]...there's no way less than a full time person would be able to keep up with getting all the requirements.*" — PNum, Agile Coach, NZ

**Key Points**: "*Difficulty in eliciting collaboration time with customers*" and "*Customer involvement in Agile demands full time involvement*"

**Code**: Lack of customer collaboration time

The codes arising out of each interview were constantly compared against the codes of the same interview, and those from other interviews and observations. This is GT's *constant comparison method* [36, 38]. In this example, other similar codes were "Insufficient time allocated to customer representatives (PNum, NZ)" and "Playing Agile customer alonside operational job (PNum, India)". Using the constant comparison method we grouped these codes to produce a higher level of abstraction, called *concepts* in GT.

**Concept**: Lack of time commitment

Other concepts that emerged include: Skepticism and Hype, Distance Factor, Dealing with Large Customers, and Ineffective Customer Representative [43]. Finally we re-

peated the constant comparison method on concepts to produce a third level of abstraction called *Categories*.

**Category**: Lack of Customer Involvement

The other concepts and categories emerged in a similar fashion. These concepts and categories form the contexts in which practitioners find it difficult to follow Agile methods by-the-book. Other set of concepts formed the adaptation strategies used to adapt Agile practices to suit these challenging contexts. For example, the adaptation strategies used to overcome the problem of *Lack of Customer Involvement* include Story Owners and Customer Proxy. Figure 1.a shows shows the levels of data abstraction using GT and 1.b illustrates the emergence of the category *Lack of Customer Involvement* from underlying concepts.

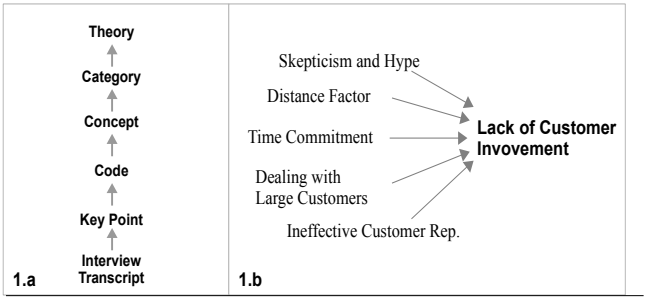


**Figure 1.** a: Levels of data abstraction in GT b: Emergence of category *Lack of Customer Involvement* from concepts

Other important elements of GT include memoing and sorting. Memoing is the ongoing process of writing theoretical memos or notes throughout the GT process [35]. Writing memos helped us pour-out all the ideas about certain codes, concepts, and categories and allowed us to capture the conceptual connections between categories [47]. Once we were nearly finished with data collection and analysis, we began to conceptually sort the theoretical memos - a process called sorting. Sorting of the memos forms a theoretical outline [47].

The final step of GT is generating a theory, also known as *theoretical coding*. Theoretical coding involves conceptualizing how the categories (and their properties) relate to each other as hypotheses to be integrated into a theory [35]. Following Glaser's recommendation, we employed theoretical coding at the later stages of analysis [34], rather than being enforced as a coding paradigm from the beginning as advocated by Strauss [76].

Glaser lists several common structures of theories known as *theoretical coding families* [35, 37]. By comparing our data with the theoretical coding families, it emerged that the coding family best 'fit' for our data was the "Strategy" family [35]. The strategy family includes strategies, tactics, mechanisms, handling, techniques etc. We have used the strategy family to describe our theory of adapting Agile practices to suit their context of use in the following section. Table 2 shows the different concepts and categories that form the *Contexts* and the corresponding concepts that form the *Adaptation Strategies* used in those contexts. Since the codes, concepts, and categories emerge directly from the data, which in turn is collected directly from the real world, the resulting theory is grounded within the context of the data [5].

### 3.2 Corroborating Case Studies

The second author conducted separate longitudinal case studies on 4 organizations in North America, over a period of 7 years:

- FIN: Financial institution, involved in securities trading. Large legacy system re-engineering.
- FACT: Very large scale manufacturing process control, for continuous operation. Real-time software embedded in a network of several hundreds of processors.
- FLY: Information processing for aerospace industry, some of it safety-critical as it is deployed in the aircraft.
- MATH: Analytical methods for security trading, derived from physics theories.

All the four organizations were already familiar with Agile methods, or some Agile practices; they had received training and guidance, done some pilot projects and thought they were ready to deploy Agile on more significant endeavours. The second part of table 1 shows participant and project details for the corroborating case studies.

The second author was brought in as a consultant when the projects hit some difficulties applying Agile methods by-the-book. These four case studies independently corroborate the findings of the Grounded Theory study.

***Data Collection*** The second author had a series of consulting and training engagement with 4 companies in diverse industrial sectors, and at various stages of agile adoption. Examining their current state, analyzing their current difficulties, and interviewing different teams members — from CEOs, program mangers and chief architects, to developers, testers and student interns — provided a great source of insight both on Agile processes themselves, and on the the impact of the context on the success of adoption. Further research is being conducted with one of the companies, to analyze the root causes of their technical debt [26, 58].

***Data Analysis*** In the case of the four consulting and training engagements, the analysis of the findings was done jointly with senior team members, and senior management to devise a plan for remediation or for going forward, plan that included some process adaptation.

### 3.3 Limitations and Threats to Validity

The inherent limitation of a Grounded Theory study is that the resulting theory can only be said to explain the specific

**Table 2.** Adaptation Strategies for Different Project Contexts

| Contexts | Adaptation Strategies |
|---|---|
| Lack of Customer Involvement | Story Owners<br>Customer Proxy<br>Simulation |
| Fixed–bid Contracts | Providing Options<br>Buffering |
| Design/Architecture Intensive | Information Architecture<br>Design Pipeline<br>Walking Skeleton |
| Documentation Intensive | Project Dictionary<br>Comprehensive Documentation |
| Slow Rate of Change | Working from Requirements |
| Distributed Teams | e-Collaboration |

contexts explored in the study. The adaptations that we discovered during the Grounded Theory part of our research are not an exhaustive list of all existing adaptations, and our study is specific to the industries of a few countries. As well as this, and as with any empirical software engineering, the very high number of variables that affect a real software engineering project may it difficult to conclusively identify the impact that any one factor has on the success or failure of the project.

In the following sections, we discuss the findings of our research studies. We do not claim our findings to be universal, rather they accurately describe the contexts studied. We have selected quotations drawn from our interviews that shed particular light on these categories and that are spread across participants, geographically and by their organizational title. The discussion is grounded further by underlying key points, codes, and concepts from our interviews and observations, that we cannot describe in detail for space reasons.

## 4. Results: Agility in Practice

In this section, we present our grounded theory: the strategies used by Agile practitioners to adapt Agile practices to their context of use. Faced with obstacles that are outside the participants' control, they have developed strategies for adapting Agile practices to their context. In the remainder of this section we present examples from our studies to illustrate when and how teams adapted Agile practices.

These adaptation strategies are summarized in table 2.

### 4.1 Lack of Customer Involvement

Most of the participants in our Grounded Theory study reported that the customer was not heavily and frequently involved in their projects (P1-P12, P14-P19, P21-P23, P25, P26, P28-P30). This lack of involvement is in sharp contrast to the amount advocated by Agile methods such as Scrum and XP. Lack of customer involvement was seen as "the most difficult part of Agile" and "the biggest problem" because "Agile [requires] fairly strong customer involvement" (P4, P17, P30). Agile teams found gathering requirements from customers as "one of the worst things" and "biggest frustration" on the projects (P8,P10). Getting customer representatives to clarify requirements is also a problem because of their unavailability:

> "*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, NZ

If the participants applied Agile methods by-the-book then insufficient or ineffective customer involvement meant that development teams were unable to gather requirements and to get customer representatives to clarify them in time for development to commence. Inability to gather requirements in time for the iterations could result in "the project get[ting] stalled" (P5) or loss of productivity:

> "*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, NZ

Without clear requirements and feedback, the teams were forced to "make more business decisions than [the team] would like" (P15) and as a result would get "misaligned from the desired business drivers" (P5) consequently requiring costly rework (P2, P4, P15). Rework is taxing for developers because they have to revisit stories developed several iterations ago due to delay in customer feedback:

> "*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, NZ

When the level of customer involvement was not enough and was outside the participants' ability to change, several teams made adaptations to their existing Agile practices in order to suit their project context. These adapted practices were: use of *Story Owners* in place of Product Owners and *Customer Proxy* from within the development team acting as a customer representative [43]. We explain the two adapted practices below.

***Story Owners*** The practice of assigning Story Owners was an adaptation to the Scrum practice of allocating a Product Owner [72]. Story owners were 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 served a three-fold purpose. Firstly, having multiple story owners instead of a single customer representative for entire project meant no one person from the customer's organization was expected to be continuously available.

> "*We didn't need that story owner for the duration of the project, we normally only need them for part of an iteration.*" – P19, Senior Agile Coach, NZ

Secondly, it allowed the team to plan out stories for development in synchronization with the corresponding story-owner's availability. Thirdly, it encouraged a sense of ownership among customer representatives as they were 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.*" — P19, Senior Agile Coach, NZ

The adapted practice of story owners proved successful for the practitioners as it helped gain the much needed customer collaboration, albeit in smaller chunks and across multiple customer representatives rather than the prescribed single Product Owner. 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.*" (P19)

***Customer Proxy*** Some Agile teams used a customer proxy — a member of the development team co-ordinating with the customers — to secure requirements and feedback. The use of proxy was visible in Indian teams where the customers were physically distant.

> "*Some customers say 'okay we know you do something called Agile and we are interested in every 6 months [involvement] but we can help you prioritize the backlog but we really don't know the terms like user stories and all'. Then we have someone called proxy customer, who really understands the customers whose there near to him, then he is able to place the view point... as long as he is able to answer questions and come back. He may not be able to answer everything off the shelf, but get the answers and get back.*" — P23, Developer, India

> "*Using Client proxy, so we assign a customer representative who interacts with the team much often but then passes on the feedback from the customer to the team and vice versa.*" — P37, Agile Coach, India

The use of a proxy to co-ordinate between the customers and the team was also observed in New Zealand, where a business analyst and couple of developers on different teams served as the proxies because of their communication skills (P2, P4, P8).

> "*We've got two people [playing proxy]...[due to] their ability to communicate ideas; they're well-spoken and able to get those ideas across...which is great for developers!*" — P13, Developer, NZ

***Simulation*** The FACT and FLY corroborating case studies had a low level of end-user involvement due to their minimal user interfaces. In comparison, the MATH case study involved physicists and mathematicians trying to apply new models and algorithms to support securities trading. The development was not driven by customers needs, but innovative work trying to find potential use. In their business model, there was no customer out there to able to express a need, rather they created new innovative tools and tried to see if they would work, by *simulation*, then find potential users. So while there was end-user involvement, this was not in the traditional Agile model of the end-user having a specific task to accomplish, describing it in the form of user stories to the development organization.

### 4.2 Fixed-Bid Contracts

The Agile Manifesto values "customer collaboration over contract negotiation" [41]. However in our Grounded Theory study we found that many of our Indian participants struggled with their customers' demand to fix time, cost and scope in a fixed-bid contract. The practitioners explained that the customers perceived the fixed-bid contract to give them predictability and control over the project schedule, cost, and deliverables. Since software development teams and their customers need legal contracts, and the market was such that customers could move to different software development companies if they wanted, this left our practitioners to handle the apparent contradiction between the customers' desire for *certainty* with their own commitment to Agile values such as *responding to change* [42].

Our participants mentioned contract negotiation as another main challenge they face in managing Agile projects.

> "*sometimes limitations are imposed by customers, like... contracts... they just want to give you scope, requirements and expect you to deliver it or they are looking for a fixed price contract....if you ask me biggest problems... one is contracts... they want three things: fixed deadline, fixed price, and fixed scope.*" - P27, Senior Management, India

A fixed-bid contract puts the development team under pressure to deliver to the fixed constraints in the contract. The negative consequences of a fixed-bid contract in an Agile project is captured in the following comment by an Agile trainer and coach who worked mostly 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 customer - the greater source of dysfunction comes from within the organi-*

*zation where the contract - fixed bid contract - is negotiated by 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 underdeliver and that I see again and again and again.*" — P32, Agile Trainer, India

Agile practitioners see fixed-bid contracts as a major limitation that the customers impose on them. Other practitioners shared their frustration over the issue of dealing with fixed time/scope/cost contracts.

"*Fixed price doesn't work well with Agile.*" - P39, Agile Coach, India

"*With Agile it's difficult to do fixed price projects. Agile talks about embracing change, can't do fixed price projects with changes coming in.*" - Practitioner P24, Developer, India

***Providing Options*** Our participants shared with us some of the strategies they used to deal with the customers' expectation of fixed-bid contracts [42]. Agile practitioners offer different contract options to customers in order to encourage them to try Agile. Practitioners P3 and P8 encouraged customers to buy a few iterations to begin with instead of signing a contract for a large project up front:

"*Most of the time...[we] sell a certain number of iterations.*" - Practitioner P27, 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.*" — P24, Developer, India

"*Try for a month - then buy more sprints.*" - P37, Agile Coach, 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 that are of more value to them.

"*...customer after seeing demo after 4th iteration realizes the features built, say the 13th feature, is not required and he needs something else...he can swap the two.*" — P24, Developer, India

The practitioners also provide the customers with a termination clause in the contract such that customers have the option to quit on a few iterations' notice.

"*...[customers are] open to suggestions to retreat after few sprints.*" – P33, Agile Coach, India

"*[Developers] start working on functionality from day one and you can add a sprint - not enter into contract for entire project - end in one sprint's notice and they [customers] can introduce change*" — P24, Developer, India

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

***Buffering*** Another practice used to adapt to the context of fixed-bid contracts was *Buffering*, which involved adding a buffer to the estimated time taken to complete a project or feature. Based on the rate of development per iteration - team velocity - as a guideline, estimates can be made about how long a particular set of requirements in a given domain will take to be developed. Then some amount of extra time was added to the estimated time as 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 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.*" — P23, Developer, India

The small amount of buffer time was important to allow the customer to 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.

### 4.3 Design/Architecture Intensive

One of our participants, a customer representative, shared their concern over using Agile methods on a design intensive project. Their project involved building an entertainment website and was driven by the front-end design. Drawing on their experiences, the customer representative noted that Agile methods are better suited to projects which are light on design and need to be adapted to suit projects which are design intensive:

> "*I was aware at the beginning that Agile is much better suited to pure development, and it needs to be twisted a bit to fit with design and IA (Information Architecture) and to run a whole end team project.*" — P18, Customer Representative, NZ

This team adapted regular Agile practices to make them fit the context of the front-end design-intensive project. These adaptations included the use of an *Information Architecture* (IA) document and *Design Pipeline* — the practice of running design activities ahead of development by one iteration.

***Information Architecture*** Our participant used an Information Architecture (IA) document in response to a web project context that was inherently front-end design-intensive. The IA document provided the intended layout of the website and also defined the set of rules associated with each of the elements on the document. The IA was able to capture the essence of functional specifications and requirement documents in one place without being overly formal or dictative. The designers and developers were able to use the IA to explicitly guide their front-end design and back-end functionality respectively.

> "*And so the IA will say here's your page... the top navigation here and it's going to have these tabs on it... here are some images. Now, a designer will take that and say actually I think these should be in different places on the page, and they may rearrange it visually, but from a here's-what's-on-the-page perspective, the IA defines all that and then says... here's rule number one and when it's clicked it gets the [audio] playing. So it's a little more towards the functional spec, it's a little towards a requirement document, and I don't really need a requirements document or a functional spec, I like it here in one place. So it's really here's a series of pieces and then there are the rules that parallel it. And a designer can look it and say here are the elements I need to create and where they go, and a developer can look at this and say ok this needs to happen with that.*" — P18, Customer Representative, NZ

***Design Pipeline*** One of our participants identified an adaption aimed at ensuring that developers did not waste substantial effort on technical matters prior to getting the front-end design right in a project whose context was skewed towards being front-end design-intensive. Their concern with following by-the-book Agile methods is encompassed in the following quote:

> "*My personal experience is that, developers get too hung up on the "how to do it", rather than "what exactly to do". So if the problem is not well defined before the developer gets it, he might do plenty of cool optimizations, make it very efficient, but in the end, it might not be what the people want... Agile doesn't depend much on initial specs — it can backfire sometimes.*" — P38, Designer, India

Their adaption was to support the design in driving the back-end functionality. For every iteration, the designs had to be ready before the development work could commence. The team addressed this by scheduling design activities a whole iteration ahead of development, a practice adaption we call *Design Pipeline*. The *Design Pipeline* involved starting up with front-end design activities dominating the zero'th iteration, with the consequence that the other developmental activities proceeded to continually follow the front-end design by one iteration.

> "*You have to run IA half or a whole sprint ahead of development. If you run them together you just waste you much time. I think you really need a sprint zero which is much more design and IA focused.*" — P18, Customer Representative, NZ

The adaptation made in response to a software architecture intensive project was *Walking Skeleton* (a term derived from Cockburn [16]).

***Walking Skeleton*** In the corroborating case studies, project FIN had made some very good initial progress with an XP+Scrum approach and a two-week iteration pace. However the project hit a wall after 6 months for lack of a sufficient focus on developing a software architecture that would scale and that would shield the various teams. Software architecture was initially perceived as Big Up Front Design (meaning: not a good thing), and proposals to make some major architectural investment were dismissed with cries of YAGNI — "you ain't gonna need it" [12] — or suggestions to defer to the "last responsible moment" [65]. The focus was initially to demo to the users and the management good progress at each iteration with user recognizable features. There was no stable architecture and there was compliance pressure from external standards such as the Sarbanes-Oxley act [81]. There was a naive expectation that the right architecture will gradually emerge out of weekly refactorings. The scope of such refactorings became larger and larger, expanding beyond the boundary of a two week iteration, and this brought the project to a complete stop. Tensions arose in the project triggering the departure of several key players, then the dismissal of a few key Agile proponents.

The project was restarted, and a robust architecture was designed, prototyped as a *Walking Skeleton* [16], and then had new code gradually ported onto it. This project was finally deployed operationally, though two years later than originally expected.

### 4.4 Documentation Intensive

While the Agile Manifesto sees value in documentation, it favours working software more [41]. Over the past decade this has led to the practice of Agile projects focusing more on the task of coding than the task of documentation, and the perception of light documentation being just enough documentation [75].

Several of our participants worked on projects in domains where light documentation was not just enough documentation. This stemmed either from regulatory need, or a communication need between our participants' teams and their customers. As well as this, in at least one case, existing process documentation was used as a tool to aid initial Agile adoption in a heavily bureaucratic government context, and to encourage stakeholders to take up other Agile practices.

> "*We've written a large amounts of documentation. One of the key misconceptions is that it is chaotic and that anything will happen and so the first thing you do is we present a very solid, very structured methodology to people with documentation backing up all aspects of it. That way they feel secure that this is not gonna be a random process, they are not gonna be thrown into a black-box and hopefully come out the other end.*" — P8, Agile Coach, NZ

In this section, we explore adaptions to regular Agile practices that usefully shifted some part of the focus back to documentation again.

***Project Dictionary*** Some of our participants encountered difficulties in collaborating with customers that was not solely related to their lack of involvement.

There is often a language gap between the development teams and the customers, since the development teams use technical language while customers are more used to business language [46]. The language gap between development teams and their customers poses a threat to effective team-customer collaboration by limiting their understanding of each other's perspectives. Customer representatives often have limited time to offer to the team for clarification and explanation of requirements and ineffective translation of business requirements into technical tasks can result in development of software that is misaligned from desired business drivers.

We found that one of the Indian teams extended their use of documentation by using a 'dictionary' to assist everyone on the team better understand business requirements [46]. This dictionary was an online editable document (wiki) populated by the customers with business terms, their meaning, and their context of use. These business terms were translated directly into code by the team using the same variable names, providing one-to-one mapping between the customers' business terms and their technical implementation for a given project. The customers were 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!*" — P40, Developer, India

***Comprehensive Documentation*** In some rare circumstances, very detailed documentation is a major deliverable of the project, and not just an accessory. For example, Project FLY involved developing software which in part was used in airlines' cockpits and was therefore subject to certification by aeronautical authorities under DO178-B [70]. The software had an extensive test harness and a very large number of tests, many of them generated by a tool, but nonetheless allowing regression testing very early in the development cycle. For Project FLY, delivering extensive documentation to external certification agencies was an integral part of the task. Newcomers to the team, familiar with Agile methods and XP in particular, did not readily understand the value of that standard, a key element in this business, and initially dismissed it. Documentation was deemed anti-Agile, and contrary to the spirit of the Agile Manifesto [41] by some new team members, and led to some internal clash inside the team, pushing back on what they perceived as the ultimate waterfall "evil". The way forward was a kind of "Agile undercover" [43] approach, where the team proceeded internally in an Agile fashion, but externally presented all required artifacts in a waterfall-like sequence. While still perceived by some as a "*totally brained damaged*" process, the project was able to progress, though it had experienced some turn-over with some Agile developers leaving in disgust.

### 4.5 Slow Rate of Change

Part of the motivation in the design and uptake of Agile practices was the need to react to constant change in software engineering projects. Agile is setup to strongly support garnering feedback and guiding the customer towards better understanding what they want and need [10], through close collaboration and frequently releasing working prototypes. The expectation is that the customer's informed and changing requirements continually support subtle changes in the project's direction, and many XP practices such as small releases and the planning game are specifically set up

to facilitate this. However, not all of our participants worked on projects where constantly changing requirements was a risk that needed mitigating. In fact in some projects the customers already had a clear idea of what they needed due to the nature of the problem and the structure of the industry the customer worked in.

As an example, an Agile team in NZ was catering to a customer from the Airlines industry. The customer requirements for the project were pretty stable from the start and there were no frequent changes in the requirements from the customer's end. The senior management had doubts about the suitability of Agile methods in such a context:

> "*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. Where there's batch systems processing data and spitting out, there's relatively less opportunity for interaction to demonstrate the outputs... I still am to some degree... [because] the next [project is] all about batch processing systems, so it'll mean paying a bit more attention to how [the team] gets [their] feedback [from the customers]*" — P15, Senior Management, NZ

Despite this, those participants who found themselves working in requirements-stable projects still valued many other Agile practices. Rather than choosing a non-Agile method, they instead found adaptations to those practices that assumed and were structured to deal with constant changes in requirements.

***Working from Requirements*** One example is a development team who, through a combination of Scrum and XP, were following an iterative and incremental development cycle. Because of the stable nature of the requirements, the team was able to collect most of the requirements from the customers in initial planning sessions. The team then had their own planning sessions where they translated those requirements into user stories. Where the customer had indicated a high priority, the user story was put into the earlier iterations and where the customers had low priority or no preference of priority, those stories were placed in later iterations. They would demonstrate incremental features of the product to the customer every month or so to collect feedback. Thus by working from requirements, the team was able to use iterative and incremental process of development despite the slow rate of change of requirements. The team also practiced the other Agile practices such as iteration and release planning, story boards, and retrospectives.

> "*I'm not aware of a lot of changes that have been requested... but like we've sort of come across this situation where we've gone 'hey, this would work much better if we do it this way'. And we've taken it back and maybe demoed what the screens would look like instead and got approval.*" — P13, Senior Developer, NZ

As the senior management later noted, the project "*had been highly successful*" (P15) and the team was able to deliver the project requirements while maintaining the iterative and incremental process of development and other Agile principles and practices.

In the corroborating case studies, the issues of safety and high availability consideration were predominate in project FACT. The factory product chain could practically only be stopped a couple of times a year and as such there the rate of change from a delivery perspective was very slow. Also most of the complicated software development centered around the implementation, test and optimization in code of complicated physics model, embedded in small processors, which did not lent themselves to be broken down in small, incremental 'user stories'. While a dialogue with plant operator and with physicists was useful, the teams on this projects found little use for the Agile practice of having an on-site customer representative to help with the project requirements. In absence of any 'user stories' per se, the team found it more useful to work from the specified requirement documentation. The team continued with iterative development and other Agile practices to implement the requirement documentation.

Project FLY similarly had only a few delivery windows, and the safety-critical aspect: compliance with DO178B standards [70], high stability of the requirements made iteration a help only internally, to give developers short term milestones and points for reflection, not as a way to get external input or feedback. This created a tension between the old timers and the proponents of Agile techniques: "why should we iterate furiously like this? This is exhausting."

### 4.6 Distributed Teams

Several Agile practices — such as XP's pair programming — are best supported in a co-located team environment. Co-location supports team collaboration, that in turn can help expose technical constraints and dependencies upfront so that these issues can be promptly resolved.

> "*when an [Information Architect] is sitting next to all of us together, she may go okay, this here would be quite cool, and if you click on this a flash thing comes up and the developer hears her saying this and goes 'Oh no! we're not using that technology, we can't do it!' But if they're in another room, the [Information Architect] will be quite far down the track from working that out, and when they sit sown with the developer and says okay this is what I thought, then the developer goes 'Ah, I can't do it'. [It could] have been picked up earlier on.*" — P18, Customer Representative, NZ

In the absence of team collaboration, individuals may make assumptions about the other parts of the system and may waste time and effort before realizing the technical constraints and dependencies.

However, several of our participants faced a reality where standard practices targeting team collaboration were either not an option, or not a practical option due to resource pressure.

Some teams were physically distributed across different cities or countries making co-location an impracticality. In these scenarios, they developed adaptations, and we now discuss these in more detail.

***e-Collaboration*** Agile teams are highly collaborative [41]. As one of the participants mentioned, the need for regular collaboration is limited in plan-based methods and can be done all at once by travelling to the other site. However, if our participants were to follow by-the-book Agile methods and insist on face-to-face collaborative sessions, then the need for regular collaboration through out the project would require a lot of travelling and more financial and resource investment that is typically available.

> "*Agile talks about collaboration - when outsourcing - collaboration between on-shore and off-shore teams - same is not true for waterfall projects because [in waterfall]...2 or 3 people stay [on other site] for 2 months - do requirements analysis for 2 months - [and] come back...[later] you'd be talking to same person over 9 months without seeing him...[or] talk for clarification...once in 3 days...[that] doesn't work with Agile. In case of Agile project - you need collaborations each iteration - [throughout] requirements, design, development...lots of travel in case of distributed Agile!*" — P24, Developer, India

In order to maintain regular collaboration across geographic distances, our Agile practitioners resorted to electronic collaboration (e-collaboration):

> "*Video conferencing becomes very important all about collaboration you should be sitting in same room as [much as] possible.*" — P24, Developer, India

E-collaboration was a popular means of regularly communicating across geographic boundaries using video/voice conferencing, phone, email and chat. For distributed Agile teams, e-collaboration was a practical adaption to the prescribed practices:

> "*Well on our project we are working like distributed Agile so the client - they also develop with us, we are like a team - separate teams so they're developing in USA and we're developing in India. We follow scrum practices and daily meetings on chat or via telephone. And wiki pages are most important in distributed Agile.*" — P31, Developer, India

Some teams were using an electronic Scrum board where the user stories and tasks could be easily tracked between the distributed teams.

> "*It's pretty good. Everyday we discuss three things: what we plan to do today, what we did yesterday, what the impediments are. If the impediments are big we take it after the standup and discuss it through Skype chat - the member who is facing the impediment... the time difference is good so you get an overlap time of about 4-5 hours. .. ..The product owner come up with the product backlog and we put it on JIRA [which] is hosted on the web so we have access to it... So we have the electronic scrum board.*" — P22, Developer, India

> "*[We collaborate] through the wiki. First part is we have a user story where we decide how we have to do it and make individual tickets or tasks according to that. We just take it up and start working.*" — P30, Developer, India

Using the electronic means of collaboration enabled distributed Agile teams to continue collaborative Agile practices such as release and iteration planning, working from the scrum board, and daily stand-ups.

## 5. Discussion: Agility in Context

In this section, we discuss the implications of our observations and generated theory, and how they relate to other research and opinion in Agile software development.

### 5.1 The Sweet Spot

Both Scrum and XP suit similar kinds of projects: a small, co-located team; an on-site or available customer representative; an emphasis on coding and testing early; and frequent feedback into updated requirements. Reifer et al., and Kruchten call this context the Agile "sweet spot" [51, 68]. The sweet spot mirrors the kinds of projects that the method designers had in mind when they constructed the first Agile methods. It is unsurprising that other projects in the sweet spot benefit from the application of methods such as Scrum and XP.

We found that our participants follow many Scrum and XP practices without adapting them. Where participants did adapt practices, we found the modification stems from the fact that the projects do not sit within this sweet spot. Other researchers have explored Agile projects outside the sweet spot, such as scaling up to larger projects [30, 53]. While we did not observe scalability-based problems, we did find several other aspects of project contexts that were significantly outside the sweet spot. This motivates the question we answer in this paper: what can projects do (indeed, what should projects do) when their contexts do not fall within Agile's sweet spot?

### 5.2 Abandon Agile Methods

If an Agile method's practices must be followed strictly, then Agile methods will only be applicable to projects within the sweet spot. Projects in contexts outside the sweet spot must presumably choose different development methods. More pragmatically, we can consider a cost-benefit analysis. Are Agile practices so interdependent and mutually reinforcing that any adaption completely undermines the Agile approach? Can projects benefit from one or more individual Agile practices, applied in isolation, perhaps within another kind of development method? Would another, perhaps more structured or more document-centric development method lead to greater benefits — or post less risk — than adapting an Agile method?

Our research participants did not abandon Agile methods and principles to address problems outside the Agile context. Clearly there is selection bias here: both the grounded theory study and the corroborating case studies were restricted to projects that at least claimed to be undertaking Agile development projects. Nevertheless, the projects we studied were able to function effectively, despite adapting some practices, and without any overall collapse of their development processes. Teams clearly derived benefits in terms of morale and productivity from adopting some Agile practices, and further benefits from the interactions between those practices they were able to apply. As most of our participants had prior experience in non-Agile projects they were in a position to judge the relative merits of different development approaches, and they preferred Agile methods to their previous methods of working. Abandoning Agile development was seen as neither a suitable nor desirable option by our participants.

While many projects fit within the Agile sweet spot, many other projects do not. In choosing to ignore Agile methods, rather than adapting methods to suit their contexts, projects outside the sweet spot will lose the benefits Agile development techniques could bring.

### 5.3 Context-Independent Practices

Although our participants adapted *some* of their method's practices, many other practices would be performed more-or-less by-the-book. We found most teams adhered to Scrum and XP practices such as iterative development (with iterations of various lengths across projects), iteration planning, testing, regular demonstrations of working software, and continuous improvement via retrospectives or reviews.

These unmodified practices tended to be inwards looking, focusing on the development team itself, rather that the team's interactions with customers and clients, e.g. their contract negotiations. We call these practices "*context-independent*", precisely because they do not depend on the project's external context. Because context-independent practices mostly involve just the team, they can be adopted with little effect upon other project stakeholders.

XP's test-first programming practice is the prime example of a context-independent practice. Indeed, test-first programming has evolved into an entire Agile (sub)method — Behaviour Driven Development — that can be applied across a wide range of projects, programming languages, and development methods, almost irrespective of the projects' contexts [32]. BDD provides significant benefit, even applied in isolation, without the surrounding infrastructure of other XP or Agile practices.

Context-independent practices show that choosing to adopt (or adapt) an Agile method is not an binary choice — just as the decision to adopt an Agile method (versus a non-Agile method) is not a binary choice. Our participants' projects aimed to perform context-independent practices by-the-book, while adapting other practices to fit the projects' contexts.

### 5.4 Context-Dependent Practices

In contrast to context-independent practices, we found some other practices are far more context-dependent. For example, release planning (including writing user stories and prioritization) becomes difficult in contexts where the customer is unwilling or unable to be highly involved with the project. Accepting changes in requirements throughout the project becomes difficult in contexts where the project is governed by a fixed-bid contract. Face-to-face collaboration is a challenge when teams are distributed.

In such situations, teams must resolve the conflict between their context and their practices. Teams have two basic tactics for achieving this: changing the context to fit the practices; or changing the practices to fit the context.

#### 5.4.1 Changing the Context

One of the great strengths of Agile methods is the way their practices reinforce each other. Test first programming supports refactoring by helping to ensure refactorings preserve correctness; refactoring supports incremental development because old components can be updated to meet new requirements; incremental development encourages a stable pace of working, which in turn allows time for test first programming. Projects that do not take on whole methodologies lock, stock, and barrel cannot get the benefits if this mutual reinforcement. This may be one reason why methodologists, coaches, and Scrum masters, criticize projects that adapt Agile methodologies, or do not adopt them *in toto*.

Context-independent practices, however, need to be adopted by more than just the core development team: release planning, or user stories as "a promise of a conversation" [22], require customers or their representative to participate along with the team — to play (as Cockburn would put it) the cooperative game [19]. Changing from fixed-bid to more Agile contracts requires legal and financial support, and goodwill, from two or more separate contracting companies. Co-locating a development team can impose significant relocation and travel expenses. Co-locating a team with their

customers could even require choosing highly-expensive in-house or on-shore development teams, rather than outsourcing to much cheaper offshore contractors. Another reason methdologists criticize adaption is to encourage projects to work with their stakeholders to pay these costs and so reap the benefits [18, 77, 79].

Unfortunately, in our study we found that projects often face contexts — business practices, geography, laws, or even fundamental mathematics — over which the participants' organizations have no authority or control. Such contexts were generally neither avoidable nor adaptable. There is a difference between encouraging teams to change contexts where that may be possible (albeit difficult); and insisting on changing contexts where that is impossible.

#### 5.4.2 Changing the Practices

In our study, we found participants tackling their own projects as best they could, devising strategies to adapt context-dependent practices to their projects' contexts. In doing so, they were deriving as much benefit from Agile practices as they could, given the control they were able to assert over their projects.

As one of our more reflective participants noted, a project's context must be understood before methods can be chosen or practices enforced:

> "*...how distributed the situation is, what's the type of technology mix that you have, how many people are involved...lots and lots of things like that...it's more a matter of adapting to the context of the client.*
>
> *You don't say 'oh no, Agile says...' No, there's no such thing as 'Agile says'. We need to find out together what works best, what gives us the best outcome in your context.*" — P17, Agile Coach, NZ

Another participant (P19) had gone so far as to design an *Agile risk assessment* questionnaire which listed a set of questions to identify potential risks inherent in a project context. The questions are based around the values and principles of the Agile Manifesto, and were used to discuss how well Agile values and principles fit the project context. The questionnaire results were then used to devise strategies to overcome those factors where the project context did not support Agile principles and values.

Such adaptations were not surprising because the fundamental principles and values of Agile development include to favour people and interaction (that make up a project context) over processes and tools (including, presumably, Agile development methods and practices). Again, the last Agile principle encourages adaption: "*At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*" Scrum explicitly supports the '*inspect and adapt*' principle [71] where teams are meant to inspect impediments in retrospective meetings and collaboratively devise innovative strategies to adapt to changing contexts. XP, too, fundamentally acknowledges problems in its implementation and promotes contextualization by recommending practitioners to "*fix XP when it breaks*" [82]

Values and principles are abstract: practices are concrete. Explicit instructions — i.e. practices — are much easier to apply than abstract principles. Advice to "write tests first" is much more straightforward, and much easier to check, than advice to "fix your methodology". We agree with Conboy and Fitzgerald [25] who argue that the inability of Agile methods to provide clear adaptation guidelines is "*similar to that of their traditional counterparts where the need for flexibility is acknowledged but not addressed.*" and that precisely this kind of adaptability "*underpins the very meaning of what it is to be Agile*."

## 6. Conclusion

Being Agile is all about having the courage to change [9].

In this paper, we have presented the results of a large-scale, international study of multiple projects using Agile practices. Our grounded theory, corroborated by four independent case studies, is that Agile teams need to change their practices to fit their projects contexts. Some practices are context-independent, primarily affecting the team itself: iterative development, testing, and retrospectives or reviews were employed unchanged by essentially all the projects we studied.

Other practices are much more context-dependent: our participants tended to adapt them as necessary to fit their projects' contexts. Where customer representatives are unavailable, teams find story owners or customer proxies. Where fixed price contracts are mandatory, teams negotiate contracts that explicitly specify development options, or allocate contingency funds to cover unforeseen changes. Where teams need to make architectural decisions that will be very hard to change, they dedicate sufficient time and resources to make those decisions "up front". Where teams need to produce documentation to gain product or process certification, or to work in multiple locations, they customise their processes to satisfy those contextual imperatives: producing documentation to satisfy regulators, or using electronic communications in conjunction with travelling for face-to-face meetings.

Returning once again to the final principle of the Agile manifesto:

> *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Agility, in this sense of Agility *in context*, cannot be evaluated simply by a tick-list of practices, or by weighing up "individuals" versus "processes", "software" versus "documentation", or even "responding to change" versus "following a plan" — even (or especially) where that plan is called

an "Agile Method". Rather, our study shows that Agile teams indeed evolve their methods and practices to be as effective as possible within their projects' contexts. We conclude with a question: to what extent should a team that does not adapt its practices appropriately — even if it is performing every practice from Scrum, XP, Lean, and DSDM "by-the-book" — really be regarded as an Agile team?

## Acknowledgments

We thank all the participants. This research is generously supported by research grants from the Agile Alliance, Scrum Alliance, and NZ BuildIT PhD scholarship.

## 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] P. Abrahamsson et al. New directions on agile methods: a comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering* Portland, Oregon, 2003.

[3] S. Adolph. What lessons can the agile community learn from a maverick fighter pilot? In *Agile 2006* 94–99, IEEE CS., Minneapolis, 2006.

[4] S. Adolph, P. Kruchten. Summary for scrutinizing Agile practices or shoot-out at process corral! In *ICSE Companion '08:* 1031–1032, ACM, New York, 2008.

[5] S. Adolph, W. Hall, and P. Kruchten. A methodological leg to stand on: lessons learned using grounded theory to study software development. In *CASCON '08:* 166–178, ACM, New York, 2008.

[6] S. Adolph, W. Hall, and P. Kruchten. Using Grounded Theory to Study the Experience of Software Development. Under review In Journal of Empirical Software Engineering, 2010.

[7] G. Allan. The Use of Grounded Theory as a Research Method: warts & all. European Conf. on Research Methodology for Business and Management Studies, 9–19, 2005.

[8] Barthélémy Dagenais et al. Moving into a new software project landscape In *International Conference on Software Engineering* ICSE, 275–284, 2010

[9] K. Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, 2000.

[10] K. Beck, I. Alexander. Point/Counterpoint *IEEE Software*, March/April 2007 IEEE, 2007

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

[12] B. Boehm. Get Ready for Agile Methods, With Care *IEEE Computer*, January 2002 IEEE, 2002

[13] J. Carver. The Impact of Background and Experience on Software Inspections *Empirical Software Engineering*, 9, 259–262, 2004.

[14] C.A. Crabtree, A.F. Norcio. Exploring Language in Software Process Elicitation: A Grounded Theory Approach In Empirical Software Engineering and Measurement (ESEM), 324–335, 2009.

[15] T. Chow, D. Cao. A survey study of critical success factors in agile software projects. J. Syst. Softw., 961–971, 2008.

[16] A. Cockburn. *Crystal clear: a human-powered methodology for small teams*. Addison-Wesley Professional, 2004.

[17] A. Cockburn. /Agile Development as the Middle Way With a Cliff On Either Side `http://alistair.cockburn.us/Agile+development+as+the+middle+way+with+a+cliff+on+either+side`, accessed on 21 March 2010

[18] A. Cockburn. Top ten ways to know you are not doing agile `http://alistair.cockburn.us/Top+ten+ways+to+know+you+are+not+doing+agile`, accessed on 23 March 2010.

[19] A. Cockburn. Agile Software Development: The Cooperative Game. 2ed. Addison-Wesley Professional, 2006.

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

[21] M. Cohn. *Succeeding with Agile: Software Development Using Scrum.* Addison-Wesley Professional, 2009.

[22] M. Cohn. *User Stories Applied: For Agile Software Development* Addison-Wesley Professional, 2004.

[23] G. Coleman and R. O'Connor. Using grounded theory to understand software process improvement: A study of Irish software product companies. *Inf. Softw. Technol.*, 49(6):654–667, 2007.

[24] K. Conboy. Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development *Information Systems Research*, 20(3): 329–354, 2009

[25] K. Conboy and B. Fitzgerald. The Views of Experts on the Current State of Agile Method Tailoring *IFIP*, 235: 217–234, 2007

[26] W. Cunningham. The WyCash portfolio management system. In *OOPSLA '92:* 29–30, ACM, Vancouver, 1992.

[27] Ward Cunningham. Almost Extreme Programming `http://www.c2.com/cgi/wiki?AlmostExtremeProgramming`, accessed 25th March 2010.

[28] DSDM. DSDM Consortium `http://www.dsdm.org/version4/2/public/`, accessed 25th March 2010.

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

[30] J. Eckstein. Scaling Agile Processes: Agile Software Development in Large Projects In *XP/Agile Universe 2002*, Springer, Berlin, 2002.

[31] Fraser, S. et al.: The Role of the Customer in Software Development: the XP Customer - Fad or Fashion? In OOPSLA, 148–150, ACM, USA, 2004.

[32] S. Freeman and N. Pryce. *Growing Object-Oriented Software, Guided by Tests* Addison Wesley, 2009.

[33] S. Georgieva and G. Allan. Best Practices in Project Management Through a Grounded Theory Lens. *Electronic Journal of Business Research Methods*, 6(1), 43–52, 2008.

[34] B. Glaser. *Basics of Grounded Theory Analysis: Emergence vs. Forcing*. Sociology Press, Mill Valley, CA, 1992.

[35] B. Glaser. *Theoretical Sensitivity*. Sociology Press, Mill Valley, CA, 1978.

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

[37] B. Glaser. *The Grounded Theory Perspective III: Theoretical Coding*. Sociology Press, Mill Valley, CA, 2005.

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

[39] Grisham, P. S., Perry, D. E. Customer relationships and Extreme Programming. In *HSSE '05*, ACM, USA, 2005.

[40] J. A. Highsmith, III. *Adaptive software development: a collaborative approach to managing complex systems*. Dorset House Publishing, New York, 2000.

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

[42] R. Hoda, J. Noble, S. Marshall. Negotiating Contracts for Agile Projects: A Practical Perspective In *XP2009*, Springer, Italy, 2009.

[43] R. Hoda, J. Noble, S. Marshall. Agile Undercover: When Customers Don't Collaborate In *XP2010*, Trondheim, 2010.

[44] R. Hoda, J. Noble, S. Marshall. Organizing Self-Organizing Teams In *International Conference on Software Engineering* (ICSE), 285–294, 2010.

[45] R. Hoda, J. Noble, S. Marshall. Balancing Acts: Walking the Agile Tightrope In *Cooperative and Human Aspects of Software Engineering* at ICSE, South Africa, 2010.

[46] R. Hoda, J. Noble, S. Marshall. How Much is Just Enough: Documentation Patterns on Agile Projects To appear In *EuroPLoP*, Germany, 2010.

[47] R. Hoda, J. Noble, S. Marshall. Balancing Self-Organizing Agile Teams: A Grounded Theory Under review In Journal of Empirical Software Engineering, 2010.

[48] M. Isham. Agile Architecture IS Possible You First Have to Believe! *Proceedings of Agile 2008*, IEEE, Toronto, 2008.

[49] A. Jackson et al. Behind the Rules: XP Experiences *Proceedings of the Agile Development Conference 2004* IEEE, 2004.

[50] Sami Jantunen Exploring Software Engineering Practices in Small and Medium-Sized Organizations *Co-operative and Human Aspects of Software Engineering* workshop at ICSE, 2010.

[51] P. Kruchten. Scaling down large projects to meet the agile sweet spot *IBM developerWorks*, 13 Aug 2004 `http://www.ibm.com/developerworks/rational/library/content/RationalEdge/aug04/5558.html`, accessed 25 March 2010.

[52] C. Larman and V.R. Basili. Iterative and Incremental Development: A brief history IEEE IEEE CS, 2003.

[53] D. Leffingwell. *Scaling Software Agility: Best Practices for Large Enterprises* Addison-Wesley Professional, 2007.

[54] L. Levine. Reflections on Software Agility and Agile Methods: Challenges, Dilemmas, and the Way ahead Engineering Institute Carnegie Mellon, 2005.

[55] G. Luck. Subclassing XP: Breaking its Rules the Right Way *Proceedings of the Agile Development Conference 2004* IEEE, 2004.

[56] A. Martin, R. Biddle, and J. Noble. The xp customer role: A grounded theory. In *AGILE2009*, Chicago, 2009. IEEE Computer Society.

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

[58] S. McConnell. Technical Debt. Software Best Practices, Construx Forum `http://forums.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx`, accessed 25th March 2010.

[59] S. C. Misra, et al. Identifying some important success factors in adopting agile software development practices. J. Syst. Softw. 82, 11, 1869–1890, 2009.

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

[61] S. Nerur, et al.: Challenges of migrating to agile methodologies. Com. ACM, 72–78, 2005.

[62] S. R. Palmer and M. Felsing. *A Practical Guide to Feature-Driven Development*. Pearson Education, 2001.

[63] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still. The impact of agile practices on communication in software development. *Empirical Softw. Engg*, 303–337, 2008.

[64] M. Poppendieck, and T. Poppendieck. *Implementing Lean Software Development–From Concept to Cash.* Addison Wesley, 2009.

[65] M. Poppendieck, and T. Poppendieck. *Lean Software Development: An Agile Toolkit.* AddisonWesley Professional, 2003.

[66] M. Poppendieck, and T. Poppendieck. *Leading Lean Software Development: Results Are not the Point.* Addison-Wesley Professional, 2009.

[67] R. Rasmussen et al. Adopting Agile in an FDA Regulated Environment Proceedings of AGILE2009 IEEE, 2009.

[68] D. Reifer et al. Scaling Agile Methods *IEEE Software*, July/August 2003 *IEEE*, 2003.

[69] D. Robey, R. Welke, D. Turk. Traditional, iterative, and component-based development: A social analysis of software development paradigms. *Inf. Technol. and Management*, 2: 53–70, 1, 2001.

[70] RTCA. RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics, Washington, DC, 1992.

[71] K. Schwaber. *The Enterprise and Scrum.* Microsoft Press, 2009.

[72] K. Schwaber, and M. Beedle. *Agile Software Development with SCRUM.* Prentice-Hall, 2002.

[73] M. Slinger, and S. Broderick. *The Software Project Manager's Bridge to Agility.* AddisonWesley Professional, 2008.

[74] J. Stapleton. *Dynamic Systems Development Method*. Addison Wesley, 1997.

[75] M. Stephens, D. Rosenber. *Extreme Programming Refactored: The Case Against XP*. Apress L.P, 2003.

[76] A. Strauss and J. Corbin. *Basics of Qualitative Research*. Sage, Newbury Park, CA, 1990.

[77] J. Sutherland. Nokia Test `jeffsutherland.com/nokiatest.pdf`, accessed 25th March 2010.

[78] J. Sutherland. Retrospective on SCRUM and Its Implementation in Five Companies. PatientKeeper, Inc., 2001.

[79] J. Sutherland, S. Downey and B. Granvik. Shock Therapy: A Bootstrap for Hyper-Productive Scrum Proceedings of *Agile2009*, ACM, 2009.

[80] J. Vlissides and K. Beck. XP Patterns Hatching Column, Interview `http://c2.com/cgi/wiki?VlissidesOnBeck`, accessed on 23 March 2010

[81] US Government. US Public Law 107 - 204 - Sarbanes-Oxley Act of 2002. US Governement Printing Office, 2002.

[82] D. Wells. Extreme Programming: A gentle introduction `http://www.extremeprogramming.org/rules/fixit.html`, accessed 25th March 2010.

[83] E. Whitworth and R. Biddle. The social nature of agile teams. In *Agile2007*, USA, 2007. IEEE Computer Society.