Documentation strategies on agile software development projects

Rashina Hoda*, James Noble and Stuart Marshall

School of Engineering and Computer Science,
Victoria University of Wellington,
P.O. Box 600, Wellington 6140, New Zealand
E-mail: rashina@ecs.vuw.ac.nz
E-mail: kjx@ecs.vuw.ac.nz
E-mail: stuart@ecs.vuw.ac.nz
*Corresponding author

Abstract: Agile software development methods and their proponents suggest 'just enough' documentation on agile projects. However, for practitioners of these methods it remains unclear how much is 'just enough' documentation. Based on a grounded theory study of 58 agile practitioners from 23 different software organisations in New Zealand and India, we found several documentation strategies being used by agile teams to overcome various challenges they faced. These documentation strategies include documenting electronic back-ups of physical paper artefacts that are prone to damage and loss; documenting change decisions by customers to trace changes and avoid disagreements; documenting business terminology for more effective requirements elicitation; documenting the traditional way when collaborating with non-agile teams; and documenting positive customer feedback to demonstrate advantage of agile adoptions. These documentation strategies collectively help define 'just enough' documentation by describing the different forms and amounts of documentation agile teams engage in.

Keywords: agile software development; documentation; self-organising teams; software engineering; grounded theory.

Reference to this paper should be made as follows: Hoda, R., Noble, J. and Marshall, S. (2012) 'Documentation strategies on agile software development projects', *Int. J. Agile and Extreme Software Development*, Vol. 1, No. 1, pp.23–37.

Biographical notes: Rashina Hoda is a Post-Doctoral Researcher and Lecturer at the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. She holds a PhD in Computer Science from Victoria University of Wellington, New Zealand, Bachelors in Computer Science Summa-Cum-Laude from Louisiana State University, USA, in 2003, and is a Certified Scrum Master (2009). Her doctoral research focused on self-organising agile teams and her research interests include software engineering, agile software development, empirical studies, project management and human computer interaction.

James Noble is a Professor and the Head of Research at the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interest centres around software design, ranging from object-orientation, aliasing, design patterns, and agile methodology, via usability and visualisation, to postmodernism and the semiotics of programming.

Stuart Marshall is a Lecturer at the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include information and software visualisation, software user interfaces on mobile devices, and agile software development.

1 Introduction

Principles of agile software development support working software over comprehensive documentation (Highsmith and Fowler, 2001). Proponents of agile methods often suggest the use of 'just enough' documentation on agile projects. What constitutes 'just enough' documentation, however, has been a topic of debate. Some people support the view that documentation on agile projects should be kept to a minimum in the traditional sense (Beck, 2007; Cohn, 2006; Jeffries, 2010). Beck (2007) seems to suggest that well-written code is its own documentation such that code should be so clearly and well-written that it should not require any extra documentation in the traditional sense. Jeffries (2010) supports adding comments to clarify code and considers unit tests and acceptance tests as executable forms of documentation. Documentation on agile projects is often seen as secondary to code and as a means to support communication (Kajko-Mattsson, 2008).

Ambler (2010) argues that agile software development does not forbid documentation altogether but does expect the practitioners to justify its need and use. Similarly, Ruping (2003) addressed this issue in his agile documentation pattern 'individual documentation requirements' by claiming that every project should define its own documentation requirements which suggests 'just enough' is context-specific.

Finally, some other groups of practitioners and researchers assert that not only will the level of documentation required on a project vary based on project contexts, but also that some amount of traditional documentation may be unavoidable in certain contexts (Hoda et al., 2010b). They go on to question the suitability and value of agile software development methods in project contexts which require comprehensive documentation efforts.

With such varying opinions and rather contradictory advices on the subject of documentation on agile projects, agile software development teams may find themselves struggling to define 'just enough' documentation in different project contexts they face. There is little empirical evidence on the different forms and amounts of documentation used on agile projects.

Based on a grounded theory study of 58 agile practitioners from 23 software development organisations in New Zealand and India, this article present some practical examples of documentation used on agile projects. These documentation strategies were being used by agile teams not just as a part of the development process but also specifically to overcome various challenges of performing agile software development. These strategies include:

- documenting electronic back-ups of physical paper artefacts
- documenting change decisions made by customers
- documenting business terminology in a project dictionary

- documenting traditional functional specifications when collaborating with non-agile teams
- documenting positive customer feedback to demonstrate advantage of agile adoptions.

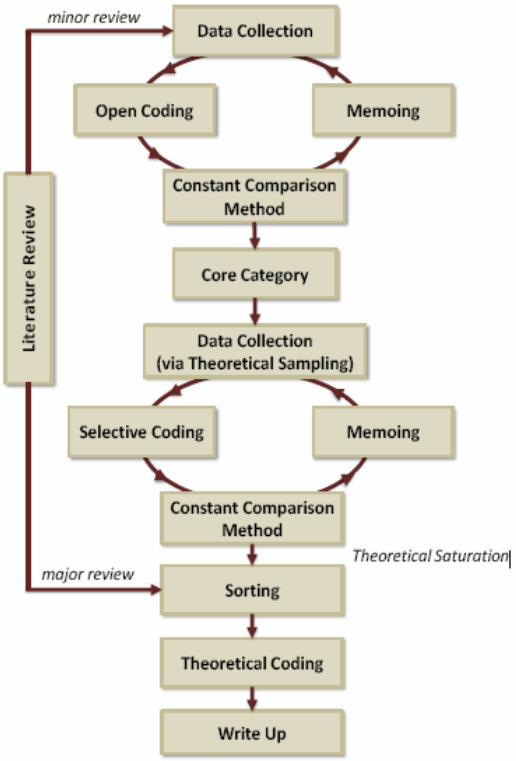
These documentation strategies collectively help define 'just enough' documentation by describing the different forms and amounts of documentation agile teams engage in on a day to day basis.

This article is an extension of our patterns paper on the same topic (Hoda et al., 2010c). The rest of the article is structured as follows: Section 2 presents our research method – grounded theory. Section 3 describes the documentation strategies. Section 4 presents a discussion of previous literature in the light of our findings. Section 5 presents some limitations of our research followed by the conclusion in Section 6.

2 Research method

Grounded theory was developed by Glaser and Strauss in 1967 as systematic method of data collection and analysis which captures a pattern of behaviour most relevant and problematic for the participants (Hoda, 2011).

Figure 1 Grounded theory life-cycle (GTLC) (see online version for colours)



Source: Hoda et al. (2011a)

Grounded theory, used as a qualitative research method, enables the study of people and interactions in order to capture the main concern of the participants and how they go about resolving it. We used grounded theory as a qualitative research method to study self-organising agile software development teams (Hoda, 2011).

Table 1 Grounded theory terms and descriptions

Term	*Description*
Minor literature review	The researcher can start off with a light literature review enough to carry on a conversation with the participants.
Theoretical sampling	A process which allows the researcher to collect, code, and analyse the data and then decide what data to collect next (Glaser, 1978).
Open coding	The first step of analysis and starts by collating key points from raw data. These are then assigned a code – a phrase that summaries the key point in two or three words (Glaser, 1978).
Constant comparison method	A process by which codes arising out of each interview are constantly compared against the codes from the same interview, and those from other interviews and observations, producing higher levels of data abstraction (Glaser, 1978).
Memoing	The ongoing process of writing theoretical notes throughout the GT process. Memos capture the conceptual links between categories as the researcher notes down their reflections on different categories.
Core category	Several categories emerge as a result of data analysis and the one that is able to account for most variations in the data and relates meaningfully and easily with other categories is called the core category (Glaser, 1978).
Selective coding	Once the core category is established, the researcher ceases open coding and uses selective coding – a procedure where they code for only the core category and those categories that are closely related to the core.
Theoretical saturation	When further data collection and analysis on a particular category leads to a point of diminishing results – no new insight into the category is generated – the category is said to have reached theoretical saturation (Glaser, 1978). The researcher can then stop collecting data and coding for that category.
Major literature review	As the theory starts to emerge, the researcher can conduct extensive literature review to see how the literature in the field relates to their emerging theory.
Sorting	Once the researcher has nearly finished data collection and coding is almost saturated, they can begin arranging the theoretical memos on a conceptual level or sorting. Sorting results in an outline of the theory describing how the different categories relate to the core category (Glaser, 1978).
Theoretical coding	Glaser lists several common structures of theories or theoretical coding families (Glaser, 1978) which can be used as a framework to describe how the categories relate to each other as hypotheses to be integrated into a theory. This is called theoretical coding.
Write up	The final step in GT is writing up the theory, which follows the theoretical outline generated as a result of sorting and theoretical coding.

Source: Hoda (2011)

Figure 1 presents an overview of the GT method or the GTLC (Hoda, 2011; Hoda et al., 2011a). The diagram captures the main procedures of the grounded theory method but does not imply a linear sequence because GT procedures are cycled and go on "simultaneously, sequentially, subsequently, serendipitously" (Glaser, 1978). The main components of the GT method are further described in Table 1.

Table 2 Participants and projects

P#	*Positions*	*Method*	*Org. size**	*Location*	*Domain*	*Team size*	*Project (months)*	*Iteration (weeks)*
P1-P9	Dev x 3, BA, AC x 2, AT, Tester, Cust. Rep.	Scrum	M	NZ	Health	7	9	2
P10	AC	Scrum and XP	L	NZ	Social services	4 to 10	3 to 12	2
P11-P18	Devx6, AC, SM	Scrum and XP	S	NZ	Environment	4 to 6	12	1
P19	SM	Scrum and XP	S	NZ	E-commerce	4	2	4
P20	AC	Scrum and XP	XL	NZ	Telecom and transportation	6 to 15	12	4
P21	Cust. Rep.	Scrum	XS	NZ	Entertainment	6 to 8	9	4
P22	AC	Scrum and XP	S	NZ	Government education	4 to 9	4	2
P23	AC	Scrum and XP	XS	NZ	Software development	8	12	1
P24-P25	Dev X 2	Scrum	XS	NZ	Software development	8 to 10	8	2
P26	AC	Scrum and XP	S	NZ	Farming	8	12	2
P27-P35	Dev x 4, AC, Tester, Sales, Manager, SM X 2	Scrum and XP	S	India	Agile software development and consultancy	5	6	2
P36-P39	AC X 4	Scrum and XP	M	India	Software development	7 to 8	3 to 6	2
P40	SM	Scrum and XP	S		CRM and finance	7 to 8	ongoing	3
P41	Designer	Scrum and XP	S	India	Web-based services	5	1	2
P42	AC	Scrum and XP	L	India	Telecom	8 to 15	3	4
P43	AT	Scrum and XP	XS	India	Agile training	7	8	2 to 4
P44-P45	Dev X 2	Scrum and XP	XS	India	Software development	4	1	1
P46-P53	Dev, BA x 2, AT, AC, KS, HR, SM	Scrum and XP	M	India	Agile software products and consultancy	15	12	1
P54	AC	Scrum and XP	M	India	Financial services	8 to 11	36	2
P55	AC	RUP	XS	Canada	Telecom	10 to 15	10 to 15	2 to 4
P56	SM	Scrum	M	USA	Oil and energy	5 to 8	12	2
P57	Cust. Rep.	Scrum and XP	M	USA	CRM and cloud computing	Variable	Variable	2 to 4
P58	AC	Scrum and XP	XS	USA	Health	Variable	Variable	2 to 4

Notes: P#: participant number; position: agile coach (AC), agile trainer (AT), developer (Dev), customer rep (cust rep), business analyst (BA), senior management (SM), knowledge strategist (KS);
*organisational size: XS < 50, S < 500, M < 5,000, L < 50,000, XL > 100,000 employees

2.1 Data collection

We collected data from 58 agile practitioners from 23 different organisations in New Zealand and India. These included practitioners in various organisational roles such as developers, testers, business analysts, agile coaches, senior managers, and customers. We supplemented the data collected from interviews with observations of agile practices. The project duration varied from 2 to 12 months and the team sizes varied from 2 to 20 people on different projects. The organisational sizes varied from 10 to 300,000 employees. Table 2 provides a summary of the projects and participants.

We conducted semi-structured interviews with open-ended questions with all the participants. We also observed workplaces, project artefacts, and agile practices like release/iteration planning and daily stand-ups for a couple of teams in India and two teams in New Zealand. Participants were practicing Scrum or a combination of Scrum and XP.

All participants were practicing fundamental agile practices such as iterative and incremental development (with varying iteration lengths), iteration/release planning, estimation and planning of user stories and tasks, testing, status report meetings (such as daily stand-up), frequent release of working software, and some form of retrospective meetings. A majority of the participants engaged in test-driven development and pair programming (on demand). In order to preserve their confidentiality, we refer to the participants by their organisational titles and country.

2.2 Data analysis

We performed data analysis using grounded theory's open coding, selective coding, and finally theoretical coding procedures (see Table 1). We conducted the data collection and analysis iteratively such that we refined interview questions to focus on the patterns emerging from the analysis. Other procedures of grounded theory such as memoing, sorting, etc. were also performed. A detailed description of the grounded theory method and its application in this study is provided elsewhere (Hoda, 2011; Hoda et al., 2011a, 2010a).

As a result of this study, we generated a grounded theory of self-organising agile teams which explains how software development teams take on informal, implicit, transient, and spontaneous roles and perform balanced practices while facing critical environment factors in order to become self-organising (Hoda, 2011). Details of how we conducted our grounded theory study, including the various procedures of the grounded theory method, are described in a doctoral thesis (Hoda, 2011).

3 Documentation strategies

In this article, we present the documentation strategies identified through our grounded theory research. The descriptions include selected quotations drawn from the interviews that shed particular light on these documentation strategies. The results are further grounded in data, codes, and concepts which we cannot present in full detail due to space concerns.

3.1 Documenting electronic back-up of paper artefacts

Agile teams make extensive use of paper artefacts to plan, display, and store their project information. These paper artefacts include user stories represented on cards, tasks on post-its, velocity or burn-down charts, etc. They serve as *information radiators* as they are often displayed in highly visible areas on story walls for the benefit of the entire team. While the importance of paper artefacts on agile projects is undeniable, they also run the risk of being easily misplaced, lost, or damaged (Hoda et al., 2010c).

A participant shared their experience of losing important project data in the form of their story board and all the paper post-its (representing the user stories and tasks):

> "All of us sitting in the same space so we didn't really bother with any formal documentation... We'd made a heck of a mess of the walls... About beginning of the fourth week, came into the office Monday morning and our walls are pristine! The cleaners had been in and they had removed everything... All the papers were gone, the story cards were gone, even the white board drawings that we'd done were gone... In retrospect, really, really funny [but] at the time: you walk in and your heart sinks... it took us 3 days just to get back to where we were." Senior Agile Coach, New Zealand

To make matters worse, the team had not documented any electronic back-ups of their paper artefacts. As a result, they had to spend a significant amount of time recalling and recreating all of their lost data. One of the strategies that they developed from this experience was to maintain regular electronic back-ups of their paper artefacts. Such back-ups can be documented in the form of digital photographs of several tasks and stories after every significant change to the story wall.

> "… at that point I realized how important are digital cameras! And from that day onwards whenever I work with a story wall I at least make a daily backup, just a photographic backup of what's up there [on the wall]." Senior Agile Coach, New Zealand

Some teams were documenting the stories and tasks by saving them electronically over servers accessible by all members of their organisation. Such form of electronic documentation also serves another important purpose: to allow distributed team members (located in different parts of the country/world) to keep up-to-date with the data otherwise easily available to the co-located team members in form of physical artefacts.

Electronic back-ups of paper artefacts become an important form of documentation on agile projects. The responsibility of documenting electronic back-ups can be rotated among team members or can be taken up entirely by the Scrum Master or Project Manager on the team. The frequency of the electronic documentation depends on the frequency of significant changes in the data stored on the artefacts. Some teams may find it useful to document on a weekly basis while others may find it sufficient to document these electronic back-ups every iteration.

3.2 Documenting change decisions

An important focus of agile software development methods is 'responding to changes' (Highsmith and Fowler, 2001) or 'embracing change' (Beck, 1999). Agile projects are often marked by high degree and frequency of changes in requirements at various stages of the project. Theoretically, customers can request changes in requirements at any time. In Scrum projects, changes are allowed to be introduced anytime before the sprint

(iterative development cycle) begins (Schwaber and Beedle, 2002). Such changes can be even more frequent if the customer is co-located with the development team such as the on-site customer in XP (Beck, 1999). One of the ways agile methods support a flexible change structure is through the use of paper artefacts to plan and store project requirements. The transient nature of these paper artefacts such as index cards or post-its is meant to support the introduction of changes in requirements without any lengthy change control process.

A challenge arising out of this arrangement is that it is difficult, if not impossible, to trace change decisions made on paper artefacts. The need for tracing change decisions may arise in the event of a disagreement between the team and their customers over particular changes. Using legal contracts or rigid change control processes to resolve such cases is counter-productive to the highly collaborative nature of agile software development.

A strategy used by one of the teams to avoid disagreements over changes and enable smooth resolution of such issues was to document all change decisions in the informal format of a wiki document. The team would use a wiki to document the change requests made by the customers with appropriate time-stamps.

> "Clients change their minds a lot, that's difficult. The answer to that is to get agreement from the client at various stages and so when you refer back to something that been agreed upon, hopefully you are not referring too far back, you're referring back to something recently that was agreed upon... transcribe into wiki. There's time stamping with wiki so you can always go back in case somebody changes it. So a lot of times when you get into disagreements with the client about features... what a feature is and isn't and whether it's in scope or out of scope. You just have to be strict about it and pointing back say... hey look two weeks ago we agreed that these story cards were gonna be done and these story cards that we are questioning now were gonna be done in a certain way and you can refer to it... I'm telling you the end result of a lot of pain of learning. If you do not do something about documenting the decisions then you get into verbal disagreements with the client and they are paying you money." Agile Coach, New Zealand

Documenting change decisions is valuable on agile projects which are typically characterised by high fluctuations in requirements. Using a wiki to document change decisions allows the team and customers to effectively trace changes while maintaining an informal and open nature of collaboration between the two parties. Such documentation helps bring in a defined structure to tracing changes while avoiding unnecessary legal complications in most cases.

3.3 Documenting business terminology into a project dictionary

Agile methods expand the customer's role in the entire process of software development by involving them in providing requirements and clarifications on requirements and providing timely feedback on developed features (Hoda et al., 2010d). Typically, self-organising agile teams are meant to closely interact and collaborate with their customers in order to elicit requirements from them instead of 'talking' to substantial requirements documents. However, there exists a language gap between the technical terminology used by the development team and the business terminology used by their business customers (Hoda et al., 2010d). An impact of this language gap can be visible when the user-stories, written in business terminology by customers, are translated into

technical tasks by the development team. Inability of the development team to understand business terminology used by their customers can result in an ineffective translation of business requirements into technical tasks. This ineffective translation can result in the development of software features that are misaligned from the desired business drivers (Hoda et al., 2010c, 2010d). An additional problem is that customer representatives often do not have sufficient time to keep clarifying their business terminology.

A strategy developed by one of the Indian teams to overcome this challenge was to document the business terminology used by their customers into a project dictionary (Hoda et al., 2010c, 2010d):

> "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!" Developer, India

Documenting business terminology in a project dictionary helps overcome the undesired consequences of the language barrier between business customers and the development team. Such a project dictionary implemented as a wiki can be used to document the business terms, their meanings, and context of use. Every project can have its own project dictionary. Ideally, the customer, who is best acquainted with the business terms should document as many business terms as possible on the wiki towards the beginning of the project with the possibility of adding and updating later (Hoda at el., 2010c). A motivation for customers to assume this responsibility is the benefit of not having to clarify business terminology to multiple team members over and over.

3.4 Documenting the traditional way for non-agile teams

Agile methods are often introduced into a software organisation through a pilot team. While the pilot team learns the agile principles, values and practices of particular agile methods, they still need to function within a primarily non-agile organisational setting. As such, agile teams often find themselves interacting and collaborating with other teams in the organisation, most of which are non-agile. A lack of traditional documentation in the form of functional specifications makes it difficult for the two teams to work together. The project information captured in the form of user stories and tasks on pieces of paper do not easily translate into traditional documentation such as functional specifications, design documentations, quarterly reports, etc., as used by non-agile teams. One of our participants disclosed how not having traditional documentation caused far reaching consequences for their pilot agile team. Not only does lack of traditional documentation inhibit effective collaboration between agile and non-agile teams, but it also influences senior management's decision to support agile teams in the long run:

> "[The] other projects within [company name], if we tried to talk to them about this or if they wanted to now anything, they'd ask about the functional specifications and we'd just simply say there is none... they didn't understand at all and [so] trying to communicate to them was quite hard... If we had a functional specification document, no matter how high level it was, it would at least allow us to communicate a lot of the functionality and ideas within our solutions to the rest of the organisation. That's probably one of the hardest

> things we've managed to do, especially when we began integrating with other projects. We were integrating two web-based solutions together, having one functional specification of how something works and then a product backlog of ideas and features, it just didn't work... when we tried to integrate with a project or when another project would want ideas about function and specification, and we couldn't give it to them, they used to get quite high-end projects that had a lot of resources, a lot of money, a lot of political sway… and since then I've found a lot of these projects contributed to the negativity of management's ideas to our projects as well. So a new project that started up, well the original idea was that they'd replace everything we'd created, so they'd start from scratch and redo everything we did, this was simply due to the fact that we couldn't give them functional specifications and they couldn't really copy our stuff and use it with theirs. So they thought they'd just scrap us!" (undisclosed participant)

A solution to this problem is to engage in some amount of traditional documentation in order to collaborate with non-agile teams (Hoda et al., 2010b).

The amount of traditional documentation agile teams need to produce depends on the nature of the project and the level of collaboration with non-agile teams. Maintaining traditional documentation (on top of agile artefacts such as story wall, velocity chart, and burn-down charts, etc.) is an overhead for the agile team. Agile teams should therefore try and keep these to a minimum (enough to continue collaboration with non-agile teams) and at the same time try and explain their non-traditional forms of documentation to the non-agile teams and help them gradually learn to extract the required information from these agile artefacts.

3.5 Documenting positive customer feedback

Senior management support is a critical factor influencing self-organising agile teams (Hoda et al., 2011b). It is crucial for a pilot agile team to gain the support of the senior management within their organisation. The pilot agile team can attempt to secure senior management support by demonstrating the success of their project. While traditional metrics of cost, time, and scope can demonstrate the success of any project, the level of customer satisfaction achieved is also an important measure of success. Agile teams collaborate closely with their customers on a regular basis. A good self-organising agile team that works diligently to satisfy their customers receives positive customer feedback. Such feedback comes in various forms, such as during demos of features at the end of iteration or in an e-mail. Agile teams are not always careful to document such positive feedback which can be used to demonstrate their effectiveness and success to gain senior management support:

> "I think that's one of the worst things with this project is that we never really looked at collecting data, collecting actual raw statistics on how much we were improving the project... we had some customer feedback and a lot of it was positive but we never really remembered the positive, we only looked at the negatives." Agile Coach, New Zealand

Securing senior management support implies not only a continued support for the pilot agile team, but also support for propagation of more agile teams in the organisation. Documenting positive customer feedback and presenting it to senior management is an important activity for agile teams, especially for pilot agile teams trying to prove the advantage of agile methods. Both agile and non-agile projects can have appropriate data

to demonstrate their success by traditional metrics. Presenting documented positive customer feedback can help agile teams demonstrate their advantage.

4 Discussion

Agile projects rely much more on tacit knowledge accumulated in the minds of the development team rather than being dependent on traditional documentation (Boehm, 2002; Highsmith, 2003; Nerur et al., 2005; Sharp et al., 2009). Lack of explicit documentation on agile projects may lead to several issues such as difficulties in tracing design and change decisions (Kajko-Mattsson, 2008; Nerur et al., 2005). The documentation strategies identified in our study help resolve some of these issues addressed in previous literature as described below.

The need to document change decisions has been acknowledged as an important activity in software development (Kajko-Mattsson, 2008; Nerur et al., 2005). Traditional documentation is seen to improve traceability of design (Nerur et al., 2005). In their study of 18 software organisations, Kajko-Mattsson (2008) found that 13 of the 18 organisations faced problems when making changes to software systems due to the inability of software developers to trace changes decisions. In the absence of formal record of decisions, the onus of remembering change decisions falls on the individual team members (Sharp and Robinson, 2004). As described in the previous sections, our study also supports the importance of tracing changes decisions, especially on agile projects which are marked by high degree of changes. The strategy of documenting change decisions on wikis with time-stamps proved to be one solution to this problem.

The use of physical paper artefacts on agile projects have been ethnographically explored by Sharp and Robinson (2004). Paper artefacts such as index cards to store requirements and estimates and story wall to capture project progress are common forms of documentation on agile projects (Sharp and Robinson, 2004). These paper artefacts are, however, prone to loss or damage and electronic alternatives have been explored (Sharp et al., 2009). Our study also identified that loss and damage are common problems faced by teams using physical paper artefacts. The documentation strategy found useful to avoid loss of information in this manner was documenting electronic back-ups of the paper artefacts on a frequent basis. The electronic back-up is done by taking digital photographs of the cards and walls or by saving their textual information in wiki format. Maintaining such electronic back-up documents on a regular basis, however, is an overhead.

Based on previous literature, Kajko-Mattsson (2008) offer some arguments in favour of traditional documentation which include better understanding and communication of software systems, ease of learning the process, higher maintainability of software systems, and improved productivity of the engineers. At the same time, they argue that previous literature suggests that lack of proper documentation is the main cause for fast quality degradation of software systems (Kajko-Mattsson, 2008). Overall, they claim that while minimum documentation on agile projects helps gain higher productivity for the team in the short-term, the long-term issues of maintainability and evolution are ignored (Kajko-Mattsson, 2008). In particular, while agile teams experience higher levels of communication and collaboration within the team, they tend to become isolated from the rest of the organisation (Karlström and Runeson, 2005). Our study suggests that lack of

traditional documentation is one of the causes for this isolation. In absence of traditional documentation, the link between the agile and non-agile teams becomes weaker.

Karlström and Runeson (2005) further suggest that agile teams must be prepared to interface with more traditional models of software development and predict that such collaborations will require some 'overheads'. In our study, we found that such overheads come in the form of traditional documentation. Agile teams need to prepare some amount of traditional documentation such as design documents and functional specifications when collaborating with non-agile teams in order to improve communication between the two.

The importance of senior management support in adoption of agile methods has been widely acknowledged (Chow and Cao, 2008; Dybå and Dingsoyr, 2008; Nerur et al., 2005; Tolfo and Wazlawick, 2008). Our study confirms that the long term success of self-organising agile teams is highly dependent on senior management support from within the organisation (Hoda, 2011; Hoda et al., 2011b). Securing senior management support, therefore, is critical for agile teams. One of the ways they can secure management support is by demonstrating the advantage of using agile methods. Documenting positive customer feedback received at various stages of the project and presenting it to senior management can help senior management appreciate the agile project from a customer satisfaction perspective.

Overall, the documentation strategies identified in our research help overcome some of the issues raised by lack of documentation on agile projects as experienced by our participants and as addressed in previous literature.

5 Implications for practice

Agile methods often suggest the use of 'just enough' documentation on agile projects. Practitioners are, however, often left wondering what constitutes 'just enough' documentation. Our study identified some documentation strategies that were being used by agile teams not just as a part of the development process but also specifically to overcome various challenges of performing agile software development. These documentation strategies collectively help define 'just enough' documentation by describing the different forms and amounts of documentation agile teams engage in on a day to day basis. We summarise the strategies and comment on their implications for practice below:

- *Documenting electronic back-ups of physical paper artefacts*. Using this strategy, agile teams can effectively manage the data captures in the paper artefacts and ensure its durability.
- *Documenting change decisions made by customers*. Paper artefacts are meant to aid changes in requirements and development. Keeping track of changes made on paper, however, is not easy. Using this strategy, agile teams can effectively keep a track of the change decisions made during their projects. Such documentation in the form of wikis is especially valuable in situations where the team and their customer may disagree on nature or extent of certain changes.

- *Documenting business terminology in a project dictionary*. Using this strategy, agile teams can build a set of business terminology commonly used by the customers along with their meanings and context of use. This information can help development teams in translating customer requirements (written in business language) into technical tasks. Project dictionaries can be shared across teams working on the same project.
- *Documenting traditional functional specifications when collaborating with non-agile teams*. Using this strategy, agile teams (especially pilot agile teams in non-agile organisations) can communicate and liaise better with non-agile teams. Some amount of traditional documentation, such as functional specifications, allows a communication link to be maintained between agile and non-agile teams. Once the agile team gets established, they can attempt to encourage non-agile teams to better understand and work with the documentation available in an agile environment.
- *Documenting positive customer feedback to demonstrate advantage of agile adoptions*. Using this strategy, agile teams can capture positive customer feedback and use that as a mechanism to demonstrate their effectiveness as a team. This strategy can be employed to secure and maintain senior management support for self-organising agile teams.

6 Limitations

A limitation of this study is that the contexts studied were dictated by the choice of research destinations, which in turn were in some ways limited by our access to them. Similarly, the selection of research participants was limited by their willingness to participate. A limitation of the grounded theory method is that the findings explain the context studied and the results are not claimed to be universal.

7 Conclusions

Proponents of various agile methods have multiple and often conflicting views on the subject of documentation on agile projects. The concept of 'just enough' documentation is meant to be context-specific and leaves the everyday practitioners of agile methods wonder how much is 'just enough' documentation in their own contexts. Based on a Grounded Theory study of 58 agile practitioners from 23 software development organisations in New Zealand and India, we have presented some practical examples of documentation used on agile projects. These documentation strategies were being used by agile teams to overcome various challenges of performing agile software development on a day to day basis. These strategies include: documenting electronic back-ups of physical paper artefacts; documenting changes decisions made by customers; documenting business terminology into a project dictionary for more effective requirements elicitation; documenting the traditional way when collaborating with non-agile teams; and documenting positive customer feedback to demonstrate advantage of agile adoptions. Most of these strategies used an informal documentation format such as a wiki, except for one where there was no clear alternative to producing some amount of traditional

documentation when collaborating with non-agile teams. These documentation strategies collectively provide empirical evidence to help define 'just enough' documentation by describing the different forms and amounts of documentation used on agile projects. Future studies could use some of these strategies and explore their use and effectiveness in different project contexts or in relation to different agile practices such as pair-programming or continuous integration.

Acknowledgements

We thank all the participants of our research. We thank BuildIT (NZ) for a PhD scholarship, the Agile Alliance (USA) for an academic grant, Software Process and Product Improvement (NZ), and the School of Engineering and Computer Science (VUW) for their financial support of this research. We also thank Dr. Philippe Kruchten (Canada) for his valuable suggestions.

References

Ambler, S. (2010) 'Agile/lean documentation strategies for agile software development', February, available at http://www.agilemodeling.com/essays/agileDocumentation.htm (accessed on 18 August 2011).

Beck, K. (1999) *Extreme Programming Explained: Embrace Change*, 1st ed., Addison-Wesley Professional, Upper Saddle River, NJ, USA.

Beck, K. (2007) 'A theory of programming', *Dr. Dobb's Journal*, 21 November, available at http://drdobbs.com/architecture-and-design/204201170 (accessed on 18 August 2011).

Boehm, B. (2002) 'Get ready for agile methods, with care', *Computer*, January, Vol. 35, No. 1, pp.64–69.

Chow, T. and Cao, D. (2008) 'A survey study of critical success factors in agile software projects', *Journal of Systems and Software*, Vol. 81, No. 6, pp.961–971.

Cohn, M. (2006) *Agile Estimating and Planning*, Pearson Education, Upper Saddle River, NJ.

Dybå, T. and Dingsoyr, T. (2008) 'Empirical studies of agile software development: a systematic review', *Inf. Softw. Technol*., Vol. 50, Nos. 9–10, pp.833–859.

Glaser, B. (1978) *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*, Sociology Press, Mill Valley, CA.

Glaser, B. and Strauss, A.L. (1967) *The Discovery of Grounded Theory*, Aldine, Chicago.

Highsmith, J and Fowler, M. (2001) 'The agile manifesto', *Software Development Magazine*, Vol. 9, No. 8, pp.29–30.

Highsmith, J. (2003) *Cutter Consortium Reports: Agile Project Management: Principles and Tools*, February, Vol. 4, p.2, Cutter Consortium, Arlington, MA.

Hoda, R. (2011) 'Self-organizing agile teams: a grounded theory', PhD thesis, Victoria University of Wellington, New Zealand.

Hoda, R., Noble, J. and Marshall, S. (2010a) 'Organizing self-organizing agile teams', in *Proceedings of the International Conference of Software Engineering (ICSE)*, Cape Town.

Hoda, R., Noble, J. and Marshall, S. (2010b) 'Agility in context', in *OOPSLA*, pp.74–88, ACM, Reno/Nevada, USA.

Hoda, R., Noble, J. and Marshall, S. (2010c) 'How much is just enough? Some documentation patterns on agile projects', in *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP)*, Hillside Group, Germany.

Hoda, R., Noble, J. and Marshall, S. (2010d) 'What language does agile speak?', in *Proceedings of the International Conference on Agile Software Development (XP)*, Trondheim, pp.387–388.

Hoda, R., Noble, J. and Marshall, S. (2011a) 'Developing a grounded theory to explain the practices of self-organizing agile teams', *Empirical Software Engineering*, Online 1st ed., pp.1–31, available at http://dx.doi.org/10.1007/s10664-011-9161-0.

Hoda, R., Noble, J. and Marshall, S. (2011b) 'Supporting self-organizing agile teams: what's senior management got to do with it?', in *Proceedings of the International Conference on Agile Software Development (XP)*, Madrid.

Jeffries, R. (2010) 'Essential XP: documentation', February, available at http://xprogramming.com/xpmag/expDocumentationInXP (accessed on 26 July 2011).

Kajko-Mattsson, M. (2008) 'Problems in agile trenches', in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08)*, ACM, New York, NY, USA, pp.111–119.

Karlström, D. and Runeson, P. (2005) 'Combining agile methods with stage-gate project management', *IEEE Software*, Vol. 22, No. 3, pp.43–49.

Nerur, S., Mahapatra, R. and Mangalaraj, G. (2005) 'Challenges of migrating to agile methodologies', *Commun. ACM*, May, Vol. 48, No. 5, pp.72–78.

Ruping, A. (2003) *Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects*, John Wiley, West Sussex, England.

Schwaber, K and Beedle, M. (2002) *Agile Software Development with SCRUM*, Prentice-Hall, Upper Saddle River, NJ.

Sharp, H, Robinson, H and Petre, M. (2009) 'The role of physical artefacts in agile software development: two complementary perspectives', *Interacting with Computers, Special issue: Enactive Interfaces*, January, Vol. 21, Nos. 1–2, pp.108–116.

Sharp, H. and Robinson, H. (2004) 'An ethnographic study of XP practice', *Empirical Software Engineering*, Vol. 9, No. 4, pp.353–375.

Tolfo, C. and Wazlawick, R.S. (2008) 'The influence of organizational culture on the adoption of extreme programming', *Journal of Systems and Software*, Vol. 81, No. 11, pp.1955–1967.