How Much is Just Enough?

Some Documentation Patterns on Agile Projects

Rashina Hoda
Victoria University of Wellington
Wellington
New Zealand
0064 463 9998
rashina@ecs.vuw.ac.nz

James Noble
Victoria University of Wellington
Wellington
New Zealand
0064 463 6736
kjx@ecs.vuw.ac.nz

Stuart Marshall
Victoria University of Wellington
Wellington
New Zealand
0064 463 6730
stuart@ecs.vuw.ac.nz

ABSTRACT

Agile methods advocate "just enough" documentation on Agile projects. Unfortunately, there are no clear guidelines on what is "just enough" documentation. We describe a set of patterns to help Agile software development teams define "just enough" in their own contexts. Teams can also employ these documentation patterns as strategies to overcome common challenges of adopting Agile methods.

Categories and Subject Descriptors

K.6.1 [Project and People Management]: Management techniques;
K.6.3 [Software Management]: Software Development Process

General Terms

Documentation, Human Factors, Theory

Keywords

Documentation, Self-organizing teams, Agile software development

1. INTRODUCTION

"Documentation is the castor oil of programming. Managers think its good for programmers. Programmers hate it!" [10]

The perceived lack of documentation on Agile projects is one of the major criticisms faced by Agile methods over the years [14]. The Agile Manifesto defines four values of which one clearly favours "working software over comprehensive documentation" [11]. Agile methods recommend light or "just enough" documentation. But what is "just enough" and who is to define it?

Beck seems to suggest that well-written code is its own documentation [2]. In other words, the code should be so clear and well-written that it needs so extra documentation in the traditional sense. Jeffries supports adding comments to code in case the code is not clear enough [9]. He argues that unit tests and acceptance tests are executable forms of documentation. Ambler is of the opinion that Agile methods don't shun documentation per se, but requires the practitioner to justify its existence [1]. He resonates the idea of "just enough documentation to get by". Ruping, in his Agile documentation pattern "individual documentation requirements", suggests each project should define its own documentation requirements [16]. In other words, defining "just enough" is context-specific.

With so many (often conflicting) opinions on what is "just enough" documentation, it is no wonder that some Agile practitioners misunderstand the phrase and receive criticism for their documentation practices on Agile projects. With a lack of clear definition and understanding of what "just enough" encompasses, Agile teams, specially new Agile teams, may run into all sorts of problems. In this paper, we present documentation patterns on Agile projects that can help Agile teams define "just enough" in their own context and use these patterns as strategies to overcome some common challenges of adopting Agile methods.

2. THE CONTEXT

Often Agile methods are adopted at organizations either as a result of managerial decisions (top-down approach) or as a result of development team's interest in Agile practices (bottom-up approach.) In either case, most Agile adoptions start with a pilot team that is established specifically to prove the feasibility and effectiveness of Agile methods. The big bang approach – declaring all teams in the organization 'Agile' at once – in contrast, is hardly feasible and mostly discouraged. The fate of Agile adoption at the organizational level, in many ways,

depends on the success of the pilot team. The pilot team can run into all sorts of challenges in becoming a self-organizing Agile team. The documentation patterns described below, can be solutions to some of these common challenges of adopting Agile.

3. OVERVIEW

This paper contains the following patterns. **Fake Documentation** describes how agile teams can provide design or analysis documentation (after the fact if necessary) to justify their designs. **Time-Stamps** track changes in that documentation, which can also include. **e-Backups** of physical artefacts. A **Project Dictionary** can help developers and other teams learn domain vocabulary. All these documents can help developers convince managers of the **Advantage Agile** offers over other development methods.

Table 1. Documentation Patterns

Pattern	Problem	Solution
Fake Documentation	How do you co-ordinate the timing of documentations produced across Agile and non-Agile projects?	Time the production of a minimal amount of traditional documentation to co-ordinate with non-Agile teams.
Time Stamp	How do you clarify who initiated a change request?	Document change decisions with timestamp on a wiki.
e-Backup	How do you avoid losing all your story cards and task post-its?	Make electronic backups of paper artifacts.
Project Dictionary	How do you translate customer's business requirements into technical tasks?	Engage your customers to document business terms, their meanings, and contexts of use into a project dictionary.
Advantage Agile	How do you demonstrate the Agile advantage over successful non-Agile projects measured using traditional metrics?	Document positive customer feedback.

4. THE PATTERN FORMAT

All patterns presented in this paper follow a consistent format which starts with a **Pattern Name**, the **Context** of use, an **Example** of an Agile team as introduced in the following section, the **Problem** that the pattern addresses, the **Forces** at play, the **Solution**, the **Consequences** of the solution that resolve the forces (shown with a positive symbol) and those that fail to resolve them (shown with a negative symbol), the **Example Resolved**, the **Known Uses** from our research [5,6,7,8], and the **Related Practices**.

5. RELATED PATTERNS

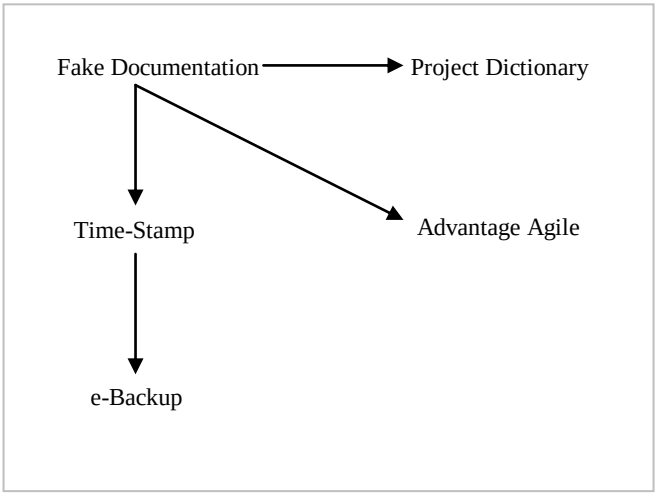

Figure 2. Related Patterns

6. SCENARIO

FlexiCo is a small software development organization catering to both in-house and external customers. FlexiCo builds web applications using the traditional software development lifecycle. One of the Project Mangers, *Alvin*, hears a buzz about a new software development methodology called 'Agile' at a local conference. He decides to investigate and ends up reading a lot of material and watching several presentations online. *Alvin* thinks his team should definitely be trying this out! So during their next meeting he introduces his team (composed of 4 developers and a tester) to the wonders of Agile software development.

*Alvin (*who insists being called Coach now by the way): "Hey folks, let's go Agile! Here are some basics..."

[two hours and 100 slides later]

Deepak, one of the developers: "(phew...no documentation) Sounds good, let's give it a shot."

Dave, another developer: "Umm don't know...(as long as we stick to this 40-hour week thing.)"

Debbie and *Derek*, the other two developers, look around to see what the others think.

Tao, the Tester: "I like the test-driven development concept (finally, I get some credit.)"

Niagara, the team lead on a non-Agile team, glances curiously from a distance: "Wonder what all this is about?"

[few weeks into the project and another 10 slides later]

Christine, the external Customer representative: "Scrum eh? I used to play Rugby. Nothing like a good rolling maul! [pause] What? I can make all the changes I want? let's do it!"

The team has been doing pretty good so far. They have delivered more or less what they promised in the past three weekly iterations. Their workplace walls are covered with story cards and post-its. They've finally started to agree on most task estimations. Just as they thought this Agile thing was working well for them, they were in for a rude shock...

7. FAKE DOCUMENTAION

Context: Co-located pilot Agile team within a non-Agile organization.

Example: By now the first and only Agile team at FlexiCo has started feeling somewhat isolated from the rest of the teams still following traditional methods. Project interdependencies, however, meant the Agile team and non-Agile teams have to exchange information. "Where are your functional specifications?" asked *Niagara*, the non-Agile team lead. "We don't do functional specs." replied *Dave*. "Yeah right" said *Niagara*. [End of conversation].

Problem: How do you co-ordinate the timing of documentations produced across Agile and non-Agile projects?

Forces

- Agile teams want to produce `just enough' documentation.
- Agile teams make (and document) decisions incrementally and iteratively, working "depth first" to produce working systems that deliver business value from the first iteration.
- Non-Agile teams make decisions according to a pre-existing, structured plan, working in a "breadth first", in the limit making decisions and implementation decisions across the whole scope of the project without producing any working code.
- Agile teams need to interface and integrate with non-Agile teams
- Non-Agile teams rely on documentation — functional specifications, design documentations, quarterly reports, etc — to record those decisions.
- Non-Agile teams perceive Agile teams' lack of traditional documentation as a weakness, and spread misconceptions and negativity about Agile teams and Agile principles.
- Such negativity can jeopardize the successful establishment and sustenance of the pilot Agile team and potential propagation of more Agile teams in the organization.

Solution: Time the production of a minimal amount of traditional documentation to co-ordinate with non-Agile teams.

The Agile team needs to interact with the non-Agile project to determine what documentation they need to produce (and when they need to produce it). The Agile team needs to schedule "story cards" to produce the documents just before they are required, as part of release and iteration planning. If the non-Agile project is well organized, most of these requirements and milestone dates will be known from inception in the project plan. After applying this pattern, the Agile team should be maintaining the minimal amount of traditional documentation necessary to talk to non-Agile teams.

Consequences:

+ Agile teams provide documentation required by their non-Agile contexts.
+ Maintaining common documentation fosters communication and understanding with non-Agile teams.
– Maintaining documentation imposes overheads that a purely Agile team would avoid — first to create the documentation, and then to update it.
– Agile teams need to resist the temptation to prioritise documentation over working software, lest they turn from the true faith and tip over the waterfall…

Example Resolved Although Dave hates any extra work (or any work most of the time) he too saw the need to maintain some amount of functional specs to hand over to Niagara, the non-Agile team lead and avoid his sarcastic sneer. Maintaining the minimal amount of functional specs meant the Agile team had some common ground to communicate and coordinate with non-Agile teams.

Known Uses: "[The] other projects within AceSqaure, if we tried to talk to them about this or if they wanted to know 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)

Related Practices:

- An example of **Fake Documentation** can be "Implied Requirements."
  [4] that are a list of functional areas and domains that can be used to describe functionality that needs to be covered.
  One of reasons behind documentation is to "support communication with an external group" [1].

  Agile Undercover [6] and the **Balancing Acts** [7] describe other techniques that Agile teams can use when their clients or surrounding projects are unwilling or unable to adopt Agile principles.

- **Time-Stamp** (next pattern)
- **Project Dictionary** (fourth pattern)
- **Advantage Agile** (last pattern)

8. TIME-STAMP

Context: Co-located Agile team.

Example: Lately, *Christine* has been asking for a lot of changes. The team, being the good Agile practitioners they are, have 'embraced' most change requests." That's the advantage of using paper, when you need to change requirements, all you do it this" said Dave, tearing up a story card. Later, at the demo of that iteration, *Christine* declares some of the changes carried out were not her decision "When exactly did I ask for that?" asked *Christine* raising a brow?

Problem: How do you clarify who initiated a change request?

Forces:

- Agile projects are marked by far more and frequent changes than non-Agile projects.
- Customers can request changes to existing software at any time — especially if they are sitting in the same room as the programmers!
- Agile teams are obliged to "embrace change". If they don't make changes immediately (or if they forget or refuse to make changes) the customer can feel cheated about Agile methods' promises.
- Agile projects heavily use paper artefacts which can be easily modified to reflect changes.
- In Agile projects, customers give requirements as 'user stories' on (paper) Story cards.
- Changes made to paper artefacts are difficult to track.
- Disagreements over change requests are not expected by teams or customers.
- Pointing to legal contracts is counter-productive in Agile projects where collaborations between the customer and the team are based on trust.
- Stakeholders may want to track and review changes while the development team may want to spend little time and effort in updating stakeholders through reports and presentations.

Solution: Document change decisions on a wiki as soon as they are made.

Use a Wiki to document change decisions. Choose a wiki engine that keeps timestamps of each edit, and the user that made the edit. Refer back to the wiki in case of disagreements over change requests. After applying this pattern, it should be clear when (and by whom) each change request was made.

Consequences:

+ The team is able to better trace project evolution over time.
+ The change wiki should be introduced to the customer as a practice meant to benefit both the team and the customer.
+ Use of wiki lends a certain 'unofficial' / Agile flavour.
+ The 'recent changes' feature on wikis can allow Stakeholders to track and review changes without the development team having to spend any extra time.
– If done too officially (such as a bound file), the log can sound 'contractual' (aka Not Agile, aka Bad).
– Maintaining common documentation can be an overhead.
– Stakeholders must accept the approach of using wikis to document change decisions.

Example Resolved: "But she asked us to change that" cried Derek and Debbie, almost in unison, at the next retrospective. "I know" said Alvin, the Agile Coach, in a calm demeanour (acquired recently after his official Scrum Master training). "It's okay. This was unexpected. Take a few minutes to reflect on possible solutions" [pause] "Let's video record all our meetings with her" suggested *Tao*, the tester, excitedly. Despite the grim air, everyone laughed. "Yeah that's a bit extreme I suppose" retreated Tao. "How about documenting all change requests?" said *Derek*. "On the wiki so there it's traceable and time-stamped" added *Debbie*. The team had found a solution. In iterations to come, *Christine*, the customer rep, found this useful too. She could now trace the progress of the projects and the evolution of the product through iterations.

Known Uses: An Agile coach shared their experience of dealing with change requests from customers:

"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 its 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 SC were gonna be done and these SC 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 u get into verbal disagreements with the client and they are paying u money." - Agile Coach, NZ

Related Practices:

- **Fake Documentation** (previous pattern)
  A negative consequence of Fake Documentation is the overhead of maintaining the common documentation. The **Time-Stamp** pattern can be to resolve the problem of how to maintain **Fake Documentation** in light of the fact that the Agile team must be able to react to change.
- **e-Backup** (next pattern)

9. E-BACKUP

Context: An Agile team relying on paper artifacts and having a "story wall" or similar exposed location for them.

Example: On returning to office one Monday, the team was shocked to find *all* the paper artefacts - their story board with story cards, tasks, and data models - had vanished! Soon enough they discovered that their project room was 'cleaned-up' by the cleaners over the weekend. Deepak shouts out in anguish while Tao holds his hand to his head: "Three weeks of project data cleaned...what a mess!" The team has no option but to rewrite all their stories, tasks, and charts from memory over the next three days. Dave was particularly grumpy that week.

Problem: How do you avoid losing all your story cards and task post-its?

Forces:

- Agile methods rely heavily on physical artifacts — such as story cards, tasks on post-its, and velocity charts — as means of communicating and storing project information.
- Physical story cards and post-its can be easily lost, damaged, or simply cleaned out.

Solution: Make electronic backups of paper artefacts.
The team should decide to make e-backup of their paper artefacts to reflect changes in their paper artefacts. Larger teams (15+ people) or those catering to projects with highly volatile requirements may find it useful to make e-backups every day, and store them into the project repository, while others might find e-backups synced to iTunes every iteration sufficient. How frequently the team makes e-backups of their paper artefacts can be decided by the answer to this question: "If we were to lose this paper artefact tomorrow, do we have enough electronic backup to replace it?" After applying this pattern, teams should be able to survive a loss of their paper artefacts (and white-board contents) needed only a few hours to rebuild their state from the backups.

Consequences:

+ Documenting e-backup of paper artefacts can help avoid unnecessary rework.
+ Electronic backup as a digital photograph of several stories and tasks at once can save effort and time.
– Electronic backup of every single physical artefact individually can be tedious and time consuming.
– Unless it is clear who is responsible for making the backups, the job can be postponed day after day…

Example Resolved: After the disaster, the team decides to maintain an e-backup of their paper artefacts. The task of making the backup is usually rotated amongst the team members. Every once in a while when no one else has time to do it, *Alvin*, the Agile coach does it. He usually takes a digital photographs of the story board, user stories and tasks at the end of each iteration. He types out the paper user stories and maintains them electronically.

Known Uses: An Agile coach in New Zealand shared the nightmarish experience of solely relying on physical artefacts on Agile projects. The absence of electronic backup led to loss of productivity and unnecessary rework:

"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 and 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 analog backup, just a photographic backup of what's up there [on the wall]." - Senior Agile Coach, NZ

Related Practices:

- The "*Super Secretary"* role [13] on a customer team can also take on the responsibility of maintaining **e-Backups** of customer artefacts. Without a Super Secretary, the responsibility can be rotated around team members.
- The Agile documentation pattern "*A Distinct Activity*" suggests documentation should be considered a distinct project activity to be assigned its own schedule, budget, and priority [16]. For larger teams with highly volatile requirements, **e-Backup** can be classified as "*A Distinct Activity*" [16] so that team members don't feel they are wasting time doing it and management understands the reason to support it.

- **Time-Stamp** (previous pattern)
  A consequence of time-stamping is overhead of documenting change decisions. However, an Agile team would already have physical artefacts that capture some of these decisions (e.g. Story cards capture requirements decisions and prioritization decisions)

and so **e-Backup** allows the team to easily capture these artefacts, the changes they represent, and the time when they took place.

10. PROJECT DICTIONARY

Context: A co-located Agile team.

Example: *Christine*, the customer representative, is getting the hang of writing user stories to express her business requirements. She reckons she's pretty good at it. Meanwhile, *Deepak*, the developer, is still struggling with the business domain terms and wonders: "how to I turn this user story into a set of technical tasks? I don't get all these business terms she's mentioned all over the story." *Deepak* attempts to clarify the user story with *Christine* but she's had enough of these identical questions from the team and tells him to go and ask Dave – or to listen when she's explaining things the first time, returning her focus to her phone conversation with her boss at the customer organisation. *Deepak* ends up making assumptions about what *Christine* really meant and creates the technical tasks related to that user story. At the next demo, while checking out the functionality *Deepak* added *Christine* utters those six unthinkable words: "This is not what I meant!"

Problem: How do you translate customer's business requirements into technical tasks?

Forces:

- Agile teams are meant to interact and collaborate directly with customers in order to elicit requirements from them instead of 'talking' to a thick requirements document with clearly defined business terms.
- Development teams speak technical language.
- Customers speak business language.
- In Agile methods, the customers are meant to provide their requirements on Story Cards written in business language which the development team needs to translate into Tasks written in technical language [8].
- Ineffective translation of business requirements into technical tasks can result in development of software that is misaligned from desired business drivers [8].
- Different business people may use different terms for the same thing — or have different meanings for the same terms [8].
- Customer representatives have limited time to offer to the team.

Solution: Engage your customer to document business terms, their meanings, and contexts of use into a project dictionary.

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. After applying this pattern, Agile teams and their customers should have a reference giving the meaning(s) of business terms and their meanings within the project.

Consequences:

+ The customer representative doesn't have to translate her business terms to technical language, which is a pain for her.
+ The customer representative doesn't have to **repeat** herself explaining terms to every team member.
+ Time spent in conversations with the customers can be better utilized to discuss the actual project rather than clarifying business terms.
+ The team can be more confident when breaking user stories down into technical tasks because they are accustomed to the business terms and contexts.
+ Good for new team members to learn the project vocabulary.
– Creating and maintaining the dictionary requires customer involvement in terms of active participation and time.

Example Resolved: *Deepak* now fully appreciates the importance of understanding business requirements correctly before development. He seeks help from his "Agile Software Development" online user group. One of the members suggests documenting a list of business domain terms and their meanings for every project. They also suggest that the best person to do it is the customer themselves since they are the subject matter experts. *Deepak* explains this to *Christine*, the customer representative, who a little hesitantly, agrees. Over time the list of terms has grown into a full-blown 'project dictionary'. The team refers to this document whenever they need clarification on business domain terms, which of course leaves *Christine* happily free from answering ad-hoc and repetitive questions about her business terms. She also noticed that the team had improved their ability to communicate with her as they frequently used business terms instead of their regular technical terms to discuss business requirements.

Known Uses: A developer shared their secret to understanding business terminology:
"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

The project dictionary was very useful for this team since they regularly practiced rotation across different project teams in a bid to increase cross-functionality.

As new members came into new projects, the project dictionary proved to be immensely useful in understanding the project's business domain and terms.

Related Practices:

- The *Super Secretary* can help create and maintain the **Project Dictionary** [13]

- **Fake Documentation** (first pattern) is an overhead. One way of mitigating that overhead is to productively use some of the documentation to build a **Project Dictionary**.

11. ADVANTAGE AGILE

Context: An Agile team in a non-Agile organisation.

Example: The project has come to an end. The team and their customer rep, *Christine*, had grown close to each other. What's more, the project was a success by traditional standards of time, scope, and price. *Sam*, the senior management, is deep in thought about the performance of the pilot team: "The project was a success. But then we've had successful projects with traditional methods. What's the advantage of doing it Agile?" *Sam*, along with other members of the management, is unable to immediately see any extra advantages of using Agile. They may decide to continue Agile for another project or they may decide to drop the idea all together.

Problem: How do you demonstrate the Agile advantage over successful non-Agile projects measured using traditional metrics?

Forces:

- Success of an Agile project cannot be demonstrated using traditional metrics such as time-scope-price.
- Traditional methods also yield successful projects — and have shelf-loads of documentation to show for their efforts.
- Agile projects may have nothing to show for themselves except working software, which the non-Agile teams also have by the end of the project.
- Demonstrating the advantage of using Agile is not easy.
- Management support is essential in the establishing and sustaining Agile teams [5]
- Most management values customer satisfaction.

Solution: Collect and record positive customer feedback.
Positive customer feedback can be documented on project wikis. After applying this pattern, the team can present that feedback to senior management, to help demonstrate the advantages of Agile approaches.

Consequences:

+ Demonstrating the advantages of using Agile methods in terms of increased customer satisfaction can help secure crucial senior management support.
– The underlying assumption, however, is that the team has indeed received positive customer feedback!
– Turning raw feedback from the wiki into a form managers can understand (Powerpoint, Word) can take some time and effort.

Example Resolved: The pilot Agile team at FlexiCo believes they have done a fantastic job. "It was both a professional and a personal growth." reflected Debbie, the developer at the team's last retrospective. "I wouldn't think of doing it any other way!" added *Deepak*, sincerely (having presented a well received session entitled "My Agile Journey" at the recent AgileNZ 2010 conference.) "I hope the management decides to continue with Agile" voiced *Tao*, the tester inviting an approving nod from the rest. "We can but hope" said *Alvin*, the Agile Coach (trying not to be mentally distracted by the couple of Agile coaching offers he had received at the last Agile professionals meet last Monday.)
[Few anxious days later] *Sam*, the senior management, announced that the team can continue to be Agile for the next project much to the relief of the team. They had learned a lot from their first Agile project and they knew better than to take this senior management support for granted. "We must really prove this [Agile] is better now so the management lets us keep it forever" touted *Dave*. "Why wouldn't they? Just ask *Christine*...she's felt the advantage." *Debbie* chipped in. "That's it! We should collect all the bouquets she has been showering on us and show it to the *Sam*!" cheered *Deepak*, in his eureka moment. "Sounds like a plan!" smiled *Alvin*. The 'pilot' Agile team at FlexiCo had 'landed' safely. "To the next iteration and beyond..." read one of the colourful posters on *Alvin's* desk.

Known Uses: The pilot Agile team at CA-NZ had gradually become efficient in their use of Agile practices. Their performance was being appreciated by positive customer feedback through different channels such as email, telephone, and face-to-face appreciation for the software. While the team was quick to note down any negative feedback and used it to improve their product, they did not document the positive customer feedback.

"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

Related Practices:

- Wikis can be used to **Time Stamp** change decisions and also to document positive customer feedback.
- Maintaining a "Review Culture" [16] will allow both the team and customers to review the contents of the wiki so that both are comfortable.
- The "ability to solve customer problems quickly" is described as the "true definition of programmer productivity" as opposed to other traditional metrics like lines of code [3].
- **Fake Documentation** (first pattern)
  One of the reasons for doing **Fake Documentation** is to ensure that the Agile team is not looked upon as a `cowboy' team as compared to the traditional documentation-abiding non-Agile teams by senior management. And so now the question that arises is why is it worth investing in Agile teams if both Agile and non-Agile teams are comparable? **Advantage Agile**!
- This technique is not specific to Agile approaches — it can be used to argue for many kinds of new practices in organizations. This pattern is often more useful to

Agile projects because they do not produce large amounts of other documentation or process quality metrics. “Fearless Change” [12] describes a range of other strategies to gain management support for organization change.

ACKNOWLEDGEMENT

We thank Kristian Elof Sørensen for his excellent shepherding of the paper. We thank Prof. Dr. Philippe Kruchten for suggesting the name of the pattern “Fake Documentation.” [15]

12. REFERENCES

[1] Agile/Lean Documentation Strategies for Agile Software Development http://www.agilemodeling.com/essays/agileDocumentation.htm (Feb 2010)

[2] Kent Beck. “A Theory of Programming”. Dr. Dobb's Journal, Nov 2007

[3] Charles Connell. “It's Not About Lines of Code” http://www.developer.com/java/other/article.php/988641/Its-Not-About-Lines-of-Code.htm

[4] James O. Coplien, Neil B. Harrison. “Organizational Patterns of Agile Software Development” Pearson Prentice Hall, 2005

[5] Rashina Hoda, James Noble, Stuart Marshall. “Organizing Self-Organizing Teams”, In the Proceedings of the International Conference on Software Engineering (ICSE), Cape Town, 2010.

[6] Rashina Hoda, James Noble, Stuart Marshall. “Balancing Acts: Walking the Agile Tightrope.” Cooperative and Human Aspects of Software Engineering (CHASE) workshop at ICSE2010, South Africa, May 2010.

[7] Rashina Hoda, James Noble, Stuart Marshall. “Agile Undercover: When Customers Don't Collaborate.” XP2010, Norway, June 2010.

[8] Rashina Hoda, James Noble, Stuart Marshall. “What Language Does Agile Speak?” XP2010, Norway, June 2010.

[9] Ron Jeffries. Essential XP: Documentation http://xprogramming.com/xpmag/expDocumentationInXP (Feb 2010)

[10] Jerry Weinberg. “Psychology of Computer Programming”

[11] The Agile Manifesto http://agilemanifesto.org/

[12] Mary Lynn Manns and Linda Rising. “Fearless Change: Patterns for Introducing New Ideas.” Addison-Wesley, 2005.

[13] Angela Martin, James Noble, Robert Biddle. “The XP Customer Roles: A Grounded Theory”. Agile2009, Chicago, August 2009.

[14] Matt Stephens, Doug Rosenber. “Extreme Programming Refactored: The Case Against XP”, Apress L.P, 2003

[15] D L Parnas, P C Clements. **“**A rational design process: How and why to fake it”, IEEE Transactions on Software Engineering, 1986

[16] Andreas Ruping. “Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects” , John Wiley, 2003