Extending Agile Methods: Postmortem Reviews as Extended Feedback

Torgeir Dingsøyr, Geir Kjetil Hanssen
SINTEF Telecom and Informatics,
7465 Trondheim, Norway
(Torgeir.Dingsoyr|Geir.K.Hanssen)@informatics.sintef.no

Abstract. Agile software development methods, such as Extreme Programming, focus on informal learning mechanisms like pair programming. Yet powerful methods, new knowledge that is gained in a project will not spread rapidly in an organisation if knowledge and experience is not externalised. We propose to combine a lightweight externalisation method: postmortem reviews with agile methods to strengthen the overall learning, and suggest how this can be done. We use practical experience from an Extreme Programming development project, and from conducting postmortem analysis in several companies in our discussion.

Keywords: Agile Development Methods, Extreme Programming, Postmortem reviews, Knowledge Management, Experience Elicitation.

1 Introduction

Agile software development methods such as extreme programming have in the past years achieved an explosive interest in the international information technology community. This can be seen as a reaction to the more traditional and control-oriented methods, where the waterfall method and its variations are examples. These methods focus on predictability through extensive planning and thoroughly follow ups on the plans. This way of running projects and develop software conflicts with the increasing need to handle rapid changes in projects and their environment. It is here that agile methods fit in. They are built to handle changes in design and requirements [1]. They open up for creativity during the whole project lifecycle and not just in the initial planning and design phases. Agile means smooth, non-bureaucratic and adaptable.

The learning processes in agile methods, and specifically extreme programming as we will discuss in this paper, are also agile. That means that the learning process is simplified compared to other more comprehensive development methodologies.

Pair programming is one of the extreme programming "practices" that are designed to support knowledge transfer within a team. This works well for pairs of programmers, but we claim that learning can be even better if this is combined with other learning modes. In this paper we suggest to extend XP with an "agile" method for harvesting knowledge from a group and documenting them in a way that makes them accessible to others: using postmortem reviews. We will show an example of a student extreme programming project where postmortems have been used.

Now, we first discuss learning in software development, and outline a framework of learning modes developed by Nonaka and Takeuchi. Then, we discuss how learning takes place now in extreme programming, before briefly discussing the research method applied here. Further, we introduce postmortem analysis, and show from a student example how this can be applied in extreme programming. We end by discussing the additional learning aspects introduced by the postmortem.

2 Learning in Software Development

There are many theories of how learning takes place in ordinary work situations. Brown and Dugid [2] have described how "communities of practise" learn from each other, David Kolb [3] describes how people learns from experience, and Nonaka and Takeuchi [4] have developed a "theory of knowledge construction".

Of course, learning also takes place in software development. New technologies, work methods as well as problems in existing software development methods requires that software developers are keen on learning.

2.1 A Model of Learning

To discuss learning in software engineering, we will rely on the model presented by Nonaka and Takeuchi. We choose this model, because of its emphasis on tacit knowledge, which shows how learning takes place in agile development methods. It is also a model that is generally known. They divide between two types of knowledge: *Tacit* knowledge, that humans have, but are unable to represent. An example is "how to ride a bike" – which is difficult if not impossible to write down. The other type of knowledge is *explicit* knowledge – which is possible to represent as process guidelines or any other form of documentation.

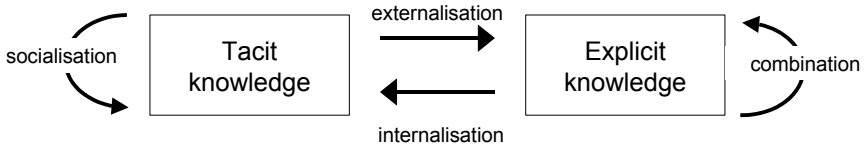

Fig. 1. Conversion of Tacit and Explicit Knowledge.

Further, Nonaka and Takeuchi divide between four modes of knowledge conversion between tacit and explicit knowledge, as shown in Figure 1:

- *Socialisation* means to transfer tacit knowledge to tacit through observation, imitation and practice, what has been referred to as "on the job" training. Craftsmanship has usually been learned in this way, where oral communication is either not used or plays a minor part.
- *Internalisation* is to take externalised knowledge and make it into individual tacit knowledge in the form of mental models or technical know-how.

"Documents and manuals facilitate the transfer of explicit knowledge to other people, thereby helping them experience the experiences of others indirectly (i.e. 're-experience' them)".

- *Externalisation* means to go from tacit knowledge to explicit. Explicit knowledge can "take the shapes of metaphors, analogies, concepts, hypotheses or models". This conversion is usually triggered by dialogue or collective reflection, but can also be the result of individual reflection, for example in a writing process.
- *Combination* is to go from explicit to explicit knowledge, that is, to combine and systematise knowledge from different sources such as documents, meetings, telephone conferences or bulletin boards. Systematising this kind of explicit knowledge is to reconfigure it by sorting, adding, combining or categorising the knowledge.

According to Nonaka and Takeuchi knowledge passes through different modes of conversion in a spiral which makes the knowledge more refined, and also spreads it across different layers in an organisation.

We will be using this framework in our discussion to come.

2.2 Learning in Extreme Programming

Extreme programming (XP) emphasises on teamwork and communication within software development teams. The originator of XP, Kent Beck, claims that the bottlenecks in software development all stem from the difficulty in communication between people [5]. One of the 13 practices in XP [1], pair programming, is designed to ensure that the programmers learn from each other. Pair programming is a technique that supports transfer of tacit knowledge, as Barry Boehm point out in an article, "agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans" [6]. This can be general programming knowledge, knowledge on the use of tools, knowledge of the application domain etc. Pair programming works as knowledge transfer as what is called socialisation in Nonakas model.

That no knowledge is documented, except in the source code, means that much relevant knowledge is inaccessible to others than the ones that work in each project.

3 Research Methods

The research reported here was carried out in a research project with participants from academia and industry, called Process Improvement for IT industry (PROFIT). Researchers work in tight co-operation with nine companies that develop software, with different process improvement initiatives.

For the part on postmortems, we have participated in collecting experience from real software projects in a real environment, which means that we have little control of the surroundings. This is often referred to as action research [7, 8]. The benefit with this type of research is that the actual problems are very relevant to industry. A difficulty

is that we have limited control over the surroundings, so the results might not be as generally applicable as when using for example experiments.

The researchers have had two roles in this work: First, as a kind of process leader who have organised a meeting: Set the agenda in co-operation with industry and organised discussions. On the other hand, the researchers have been observers trying to document the process and the results of the meeting.

For the part on extreme programming, we have run a student project using extreme programming, where four students have been working in couples for 12 weeks to develop software. We have had weekly meetings with the students to discuss progress, and have also measured progress from week to week. Our analysis and description in this paper are based on discussions with these students in addition to the literature.

4 Learning through Postmortem Reviews

A postmortem review is a review done in the end of a project to sum up good and bad experiences. There are several ways to perform such [9]. Apple has used a method [10] which includes designing a project survey, collecting objective project information, conducting a debriefing meeting, a "project history day" and finally publishing the results. At Microsoft they put much effort into writing "postmortem reports". These contain discussion on "what worked well in the last project, what did not work well, and what the group should do to improve in the next project" [11]. The size of the resulting documents are quite large, "groups generally take three to six months to put a postmortem document together. The documents have ranged from under 10 to more than 100 pages, and have tended to grow in length".

A problem with these approaches is that they are made for very large companies, who can spend a lot of resources on analysing completed projects. We work with medium-sized companies where 5-10 people usually participate in a project, ranging in size from about 8 to 50 man-months. To suit this type of projects, we have developed a "lightweight" version of postmortem reviews [12, 13].

We have used postmortem reviews as a group process, where most of the work is done in one meeting lasting only half a day. We get as many as possible of the people who have been working in the project to participate, together with two researchers, one in charge of the postmortem process, the other acting as a secretary. The goal of this meeting is to collect information from the participants, make them discuss the way the project was carried out, and also to analyse causes for why things worked out well or did not work out.

Our "requirements" for this process is that it should not take much time for the project team to participate, and it should document the most important experience from the project, together with an analysis of this experience.

4.1 Lightweight Postmortem Review

We have used two techniques to carry out lightweight postmortem reviews. For a focused brainstorm on what happened in the project, we used a technique named after

a Japanese ethnologist, Jiro Kawakita [14] – called "the KJ Method". For each of these sessions, we give the participants a set of post-it notes, and ask them to write one "issue" on each. We usually hand out between three and five notes per person, depending on the number of participants. After some minutes, we ask one of them to attach one note to a whiteboard and say why this issue was important. Then the next person would present a note and so on until all the notes are on the whiteboard. The notes are then grouped, and each group is given a new name.

We use a technique for Root Cause Analysis, called Ishikawa or fishbone-diagrams to analyse the causes of important issues. We draw an arrow on a whiteboard indicating the issue being discussed, and attach other arrows to this one like in a fishbone with issues the participants think cause the first issue. Sometimes, we also think about what was the underlying reasons for some of the main causes and attach those as well.

4.2 Example Results from a Lightweight Postmortem Review

One result from a KJ session in a satellite software company was two post-it notes grouped together and named "changing requirements", with the following statements from two developers:

"Another thing was changes of requirements during the project: from my point of view – who implemented things, it was difficult to decide: when are the requirements changed so much that things have to be made from scratch? Some wrong decisions were taken that reduced the quality of the software".

"Unclear customer requirements – which made us use a lot of time in discussions and meetings with the customer to get things right, which made us spend a lot of time because the customer did not do good enough work."

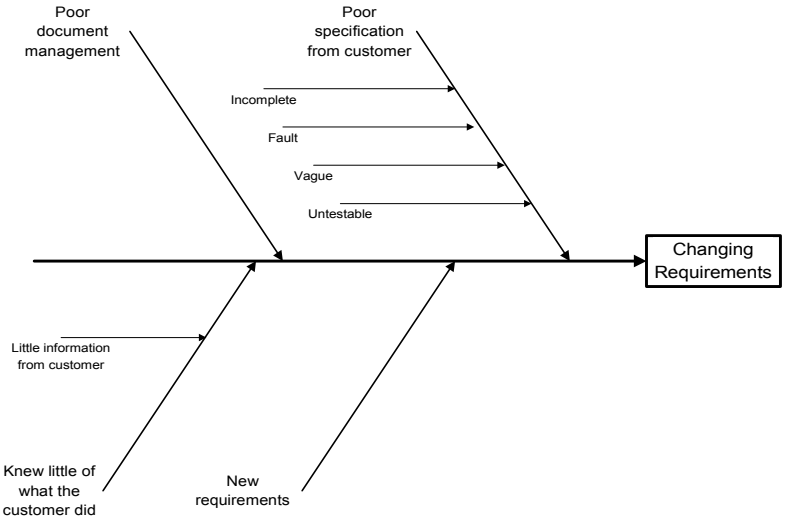

Fig. 2. Ishikawa diagram for "Changing Requirements".

When we later brought this up again and tried to find some of the root causes for "changing requirements", we ended up with the fishbone diagram in Fig. 2.
The root causes for the changing requirements, as the people participating in the analysis saw it, was that the requirements were poorly specified by the customer, there were "new requirements" during the project, and the company knew little of what the customer was doing. Another reason for this problem was that documents related to requirements were managed poorly within the company. In Fig. 2, we have also listed some subcauses.

4.3 Experience with Postmortem Reviews in Extreme Programming

Two students used postmortem reviews as a learning mechanism in XP. They developed an Intranet-based electronic process guide for a local company, and used XP as their development process. The project lasted for three months and the two programmers used approximately 430 hours on the development project. The project had 7 iterations, and all XP practices were followed. In addition to the project work they also spent approximately 20 hours on postmortem activities to collect and discuss experiences.

Initially the programmers used XP as it is described in the literature [15]. Although XP is built on many well known concepts, it needs local adaptation and tailoring [5]. With this in mind the students wanted to adapt their use of XP during the project, based on their own experiences. The practises were revised from iteration to iteration. After each of the seven iterations a postmortem was conducted to identify both positive and negative experiences. The participants were the two programmers and the customer representative; the whole session lasted for about one hour. Examples of positive experiences collected from such postmortems were *"spiking on new technology is useful"* and *"refactoring is useful in each iteration"*. Examples of negative experiences were *"get no production code when one programmer is sick (inactive)"* and *"difficult to estimate large stories"*. The programmers used these experiences to suggest improvements in the XP-practices. An example of one such improvement suggestion is *"must split large stories into small ones, maximum 1/2 day"*.

Initially each practice was printed on a paper and taped to a wall; each sheet had a description of one practice. The most interesting post-it notes (the experiences) from the postmortem were placed beside the sheet describing the related practice. In the same way, each improvement suggestion was written on a post-it note and placed beside the related practice. The programmers also wrote suggestions during the iteration. When the programmers decided to implement a suggestion, they removed the suggestion post-it note and adjusted the practice by rewriting the practice description, print it, and then place it on top of the old sheet. In this way they had a continuously experience based adaptation of XP. At the end of the project they had all changes placed on the wall and they could look back on the improvement history of each practice by looking at the old sheets below the prevailing one on the top. In Figure 3, we show how the "improvement wall" looked like during the project.

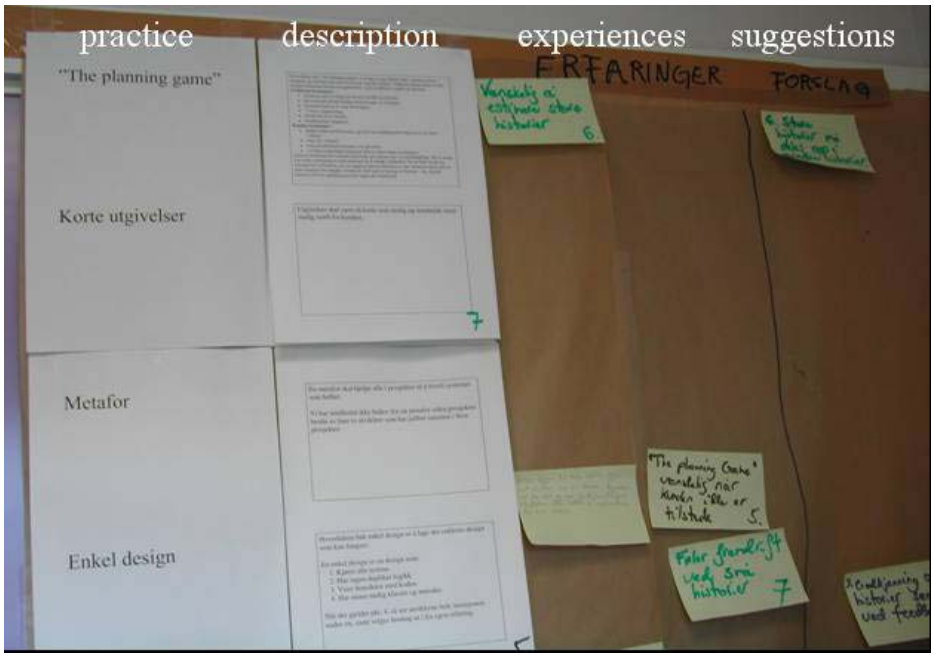

Fig. 3. The improvement wall, showing practice names and descriptions, experience and suggestions for changes (picture text in Norwegian).

Figure 4 shows the number of suggestions that came up from the postmortem on each iteration. Most of these suggestions were implemented in the next iteration.

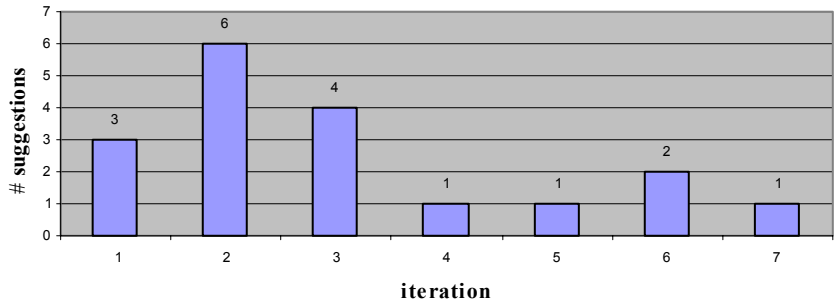

Fig. 4. Improvement suggestions per iteration.

5 Discussion and Conclusion

This paper shows how PMA and especially the KJ-technique can be used to adapt XP during a project. It is likely to believe that some of these experiences and the

suggestions from the student project never would have been found without this support. One advantage of using KJ in this case was that the programmers could evaluate their experiences in co-operation with other stakeholders in the project, which in this case was the customer representative. Other stakeholders like system architects, document and database-responsible can also be involved to a greater extent.

We have now shown postmortem reviews as a method for capturing experience from software projects, and have shown results from applying it in a student extreme programming project.

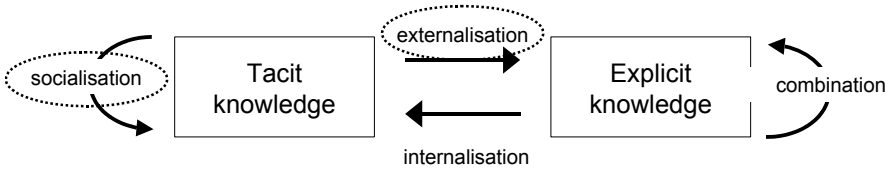

Fig. 5. Extending extreme programming for learning with both socialisation and externalisation.

Pair programming seems to be an excellent method for socialisation in software development projects, but we think that learning gets even better if this method is combined with others, as shown in Figure 5. We have shown how extreme programming can be extended in a quite "agile" way to also support externalisation.

This extension increases learning in the following ways:

- Other people than the ones participating in an XP project can gain knowledge on XP, and especially how XP is best tailored to an organisation's needs.
- The people participating in an XP project will gain a deeper knowledge of their own XP practise by externalising knowledge – and can more easily discuss changes in practise.

We think this is a method that can give new insights on local adaptations of software development, give better educated software engineers, and make XP more varied and fun for software developers.

6 Acknowledgements

We would like to thank Nils Brede Moe from SINTEF Telecom and Informatics and Tor Stålhane from the Norwegian University of Science and Technology for setting up student experiments on extreme programming. We would further like to thank the students Håvard Julsrud Hauge, Øivind Mollan, Ole Johan Lefstad, Sverre Stornes and Jørgen Brudal Sandnes, all from the Norwegian University of Science and Technology, for discussions and documented experience on extreme programming practise.

References

[1] K. Beck, "Embracing Change with Extreme Programming," *IEEE Computer*, pp. 70 - 77, October, 1999.

[2] J. S. Brown and P. Duguid, *The Social Life of Information*. Boston, Massachusets: Harvard Business School Press, 2000.

[3] D. Kolb, *Experiental Learning: Experience as the Source of Learning and Development*: Prentice Hall, 1984.

[4] I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company*: Oxford University Press, 1995.

[5] K. Beck, "Extreme Programming: A Humanistic Discipline in Software Development," presented at Fundamental Approaches to Software Engineering. First International Conference (FASE'98), 1998.

[6] B. Boehm, "Get Ready for Agile Methods, with Care," *IEEE Computer*, pp. 64 - 69, January, 2002.

[7] D. J. Greenwood and M. Levin, *Introduction to Action Research*: Sage Publications, 1998.

[8] D. Avison, F. Lau, M. Myers, and P. A. Nielsen, "Action Research," *Communications of the ACM*, vol. 42, pp. 94-97, 1, 1999.

[9] A. Birk, T. Dingsøyr, and T. Stålhane, "Postmortem: Never leave a project without it," *IEEE Software, special issue on knowledge management in software engineering*, 2002.

[10] B. Collier, T. DeMarco, and P. Fearey, "A Defined Process For Project Post Mortem Review," *IEEE Software*, vol. 13, pp. 65-72, 4, 1996.

[11] M. A. Cusomano and R. W. Selby, *Microsoft Secrets - How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*: The Free Press, 1995.

[12] T. Stålhane, T. Dingsøyr, N. B. Moe, and G. K. Hanssen, "Post Mortem - An Assessment of Two Approaches," presented at EuroSPI, Limrerick, Ireland, 2001.

[13] T. Dingsøyr, N. B. Moe, and Ø. Nytrø, "Augmenting Experience Reports with Lightweight Postmortem Reviews," presented at Third International Conference on Product Focused Software Process Improvement, Kaiserslautern, Germany, 2001.

[14] R. Scupin, "The KJ Method: A Technique for Analyzing Data Derived from Japanese ethnology," *Human Organization*, vol. 56, pp. 233-237, 2, 1997.

[15] K. Beck, *Extreme Programming Explained*: Addison-Wesley, 2000.