Augmenting Experience Reports with Lightweight Postmortem Reviews

Torgeir Dingsøyr[1], Nils Brede Moe[2] and Øystein Nytrø[1,2]
[1]Department of Computer and Information Science,
Norwegian University of Science and Technology
dingsoyr@idi.ntnu.no

[2]SINTEF Telecom and Informatics,
7491 Trondheim, Norway
(Nils.B.Moe|Oystein.Nytro)@informatics.sintef.no

Abstract. Many small and medium-sized companies that develop software experience the same problems repeatedly, and have few systems in place to learn from their own mistakes as well as their own successes. Here, we propose a lightweight method to collect experience from completed software projects, and compare the results of this method to more widely applied experience reports. We find that the new method captures more information about core processes related to software development in contrast to experience reports that focus more on management processes.

1 Introduction

Many small and medium-sized companies that develop software seem to experience the same problems in several projects, like using more effort than originally planned, for example in the test phase. To reduce the impact of such problems, project managers would often like to know what preventive actions other projects in a similar situation has taken, and what the results of these actions has been. Other projects might experience technical problems with, say a compiler, that they know have appeared in the company before, but it is difficult to find which people were involved in solving the problem then. Very few companies have systems that will capture and share this type of information.

Another characteristic of small and medium-sized companies that develop software is that they are usually under strict time pressure. They do not have the time to invest in prevention of possible future problems in the project. Usually, projects are also pretty small, involving typically between 5-10 people, which also means that they cannot use a lot of resources on this.

Here we suggest a *lightweight* method to capture experiences from completed software projects, that is suitable for companies focusing on learning from their own experience. The reasons can be either to improve the quality of their products, be more efficient, or to make the company a stimulating place to work. This is often referred to as *knowledge management* [1], and involves collecting, refining, packaging and distributing knowledge within a company.

Within software engineering, to focus on knowledge management has often been called to “set up an experience factory” [2]; an organizational framework for capturing and reusing experience from accomplished software projects. The idea here is to allocate resources for managing company-internal knowledge (the “experience factory”). –This unit should collect experience from ongoing and completed projects and make this available for others. It does not have to be a separate part of an organization, but some people should have a responsibility for it.

An interesting question in knowledge management in general, and particularly within software engineering, is how to collect, "harvest", or "make explicit" experience from projects so that they can be usable for others. What efficient methods exist, that does not require a lot of effort to make requisite knowledge available? In this paper, we are interested in looking at different lightweight approaches to capturing experience from projects, and in particular projects that are completed. This can be a way to make *tacit* knowledge [3] present in software development project *explicit,* and to store it in a knowledge repository or "experience base" to make it available as support for future projects. We will look at two different methods for capturing experience:

- *Writing Experience Reports*, reports written by project managers to sum up experience from the projects (single-author documents)
- *Conducting Postmortem Reviews*, a group process to investigate problems and successes in a project. We have developed a particular lightweight method, which does not require much effort.

To see differences between the methods, we have examined resulting reports from two software companies. Now, we first discuss the nature of experience, before saying what we mean by Experience Reports, Postmortem Reviews, and in particular *lightweight Postmortem Reviews.* We then give some context by describing the research project where this work was performed, as well as the companies where we collected the experience. Next, we limit the scope of this paper, and go on to present the research method applied here in section 2, results from each method in section 3, discuss them in section 4 and conclude in section 5.

1.1 What is experience, and what forms does it take?

Let us start by defining a related, but more broad term, "knowledge". Here, we will use "knowledge" in a quite wide sense, meaning information that is "operational", that is, usable in some situation.

Experience in a strict sense is something that resides in humans, and that is not possible to transfer to others, because you have to experience it yourself to call it an experience. We will use the word in a less strict sense here, as "a description of an event that happened during project execution". An example is "Because of frequent changes of the requirements, it was hard to see what consequences they would have. This affected the testing of the software." Each such description, we will refer to as an "experience item".

A way to categorize experience or knowledge is to look at where it is applicable, and how easy it is to transfer. Novins and Armstrong [4] have suggested the following framework for categorizing knowledge: Experience that is collected can be used in a setting that we can divide into two categories: *local* and *global*. If experience is usable only locally, it is not applicable for many people or projects. If it is globally usable, many people can benefit from it. We can also use two categories of how transferable knowledge is. If it is easily transferable, we say that it is *programmable*. If it is difficult to transfer, we say that it is *unique*. Then we get the Tab. 1.

Yet another way to classify experience would be according to the topic they are about, for example to which part of the development process they are related, to which organizational role, or to what tools or special work methods. We developed one set of categories that are relevant to the projects we will describe later:

Tab. 1. *Categorization of experience according to applicability and transferability.*

	Local	Global
Programmable	*Easy to transfer, but suits only in some situations.*	*Easy to transfer, and usable in many situations.*
Unique	*Hard to transfer, and only relevant in some situations.*	*Hard to transfer, but relevant in many situations.*

- Processes: Contract negotiation, estimation, planning, specification, design, implementation, testing, administration and maintenance.
- Actors: Customer, Project Manager, Management, Developer
- Technology: (no subcategories).

We will come back to how we applied these categorization frameworks in the discussion in section 4.

1.2 Collecting experience from projects

Now, how are we supposed to collect experience from completed projects? We first give an overview of a frequently used method, then introduce Postmortem Reviews, and explain how the effort in conducting such can be reduced to make "lightweight Postmortem Reviews".

1.2.1 Experience Reports

A way to collect experience from a completed project is to write an "Experience Report". This document is usually written by the project manager after the project is finished. The report follows a fixed template, which makes it possible to compare reports from different projects. In one of the companies we worked with, the report is divided into two parts: The first part gives an overview of all the facts and numbers from the project: Start and finish date, size of contract, labour used, deviation from estimated work size, the number of source lines-of-code developed, documents produced, and the activities that contributed most to the excess consumption. The second part of the report describes problems during project execution with proposal for improvement. For each phase of the project there is a Problem Description and Proposal for Improvements. This information is represented as text. In the other company they only have part two with problem definitions and proposed improvements.
These reports are usually not longer than 10-15 pages in one company we worked with, and about 4 pages in the other. About 50% is devoted to each part.

1.2.2 Postmortem Reviews

There are several ways to perform Postmortem Reviews. Apple has used a method [5] 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 also put quite much effort into writing "Postmortem reports", which are a bit more similar to what we have called "Experience 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" [6]. 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".

In a book about team software development, Watts Humphrey suggests a way to do postmortems to “learn what went right and wrong, and to see how to do the job better the next time” [7].
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 manmonths. To suit this type of projects, we have developed a “lightweight” version of Postmortem Reviews.

1.2.3 Lightweight Postmortem Reviews

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 try to 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. A further description of what we did can be found in the “results” section.
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.
A description of another lightweight approach which seeks to elicit experience using interviews, and not a group process, is described by Schneider [8].

1.3 The Research Setting

Here we describe in what setting the research was carried out. We first introduce the research project we worked in, and then give a brief description of each of the companies and projects where we collected the empirical data for this paper.

1.3.1 The Research Project

The Process Improvement for IT industry (PROFIT) project is a Norwegian research project where researchers are working in tight cooperation with nine companies that develop software, with different process improvement initiatives. There are three main work packages: Process improvement under uncertainty and change, learning from experience, and process improvement through innovative technology and organization. The work which is reported here focuses on learning from experience. Some background information on earlier work on knowledge management systems within Norwegian software development companies has been published earlier [9].

1.3.2 Northern Software

Northern Software makes software and hardware for receiving stations for data from meteorological and Earth observation satellites. Since the company was founded in 1984, they have delivered turnkey ground station systems, consultancy, feasibility studies, system engineering, training, and support. Northern Software has been working with large development projects, both as a prime contractor and as a subcontractor. They possess a stable and highly skilled staff, many with masters degrees in Computer Science, Mathematics of Physics, and have what we can describe as a “engineering culture”. Approximately 60 people are working in the company, and the majority is working with software development. Projects are managed in accordance with quality routines fulfilling the European Space Agency PSS-05 standards, and are usually fixed price projects.

1.3.3 Southern Software

Southern Software is a software house specialising in advanced web-solutions, but not eBusiness. Examples are games, newspapers, customer services and resources, database access etc. Projects are usually small to medium sized. Larger projects are broken down in smaller packages by incrementally adding services to a web portal.

They have a heterogeneous staff; people with software as well as design background. Project teams often consist of many people with non-overlapping competence of webdesign- and programming, user interfaces, databases and transactions, software architecture, requirements and management. This means that communication costs are fairly high compared to overall project size.

The company recently started using Rational Unified Process (RUP) for some project parts (user requirements, management and testing). They recently assigned one full-time “knowledge manager” who receives all “knowledge harvest”-documents, user surveys and other project documents. This work has just started.

2 Research Methods

The research performed is done in close collaboration with industry. 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 [10, 11]. 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 material collected for this research is from two companies that we co-operated with in the PROFIT project. They were not selected arbitrarily; they were interested in working with the same topics that we were interested in. The projects we used as cases for lightweight Postmortem Reviews were selected by the companies. However, we asked them to select “typical” projects from their company, and also projects of a certain size.

The researchers have had two roles in this work: First, as a kind of process leader who have organized a meeting: Set the agenda in co-operation with industry and organized discussions. On the other hand, the researchers have been observers trying to document the process and the results of the meeting. In one company by using a tape recorder and transcribing important sections of the meeting, in another company by writing detailed minutes during the meeting.

When we analysed the material gathered, we had two reports, one experience report, and one post mortem review report from each company. An example part of an experience report is:

In this project the team members did not sit together, which complicated the communication between the members. In the last week of the project, however, the system track was placed together, which resulted in good communication and a stronger feeling of belonging to a team.

In our analysis, we would select sentences like the one underlined, and call them experience items. This was used in our later analysis as:

negative experience: physically separate

Another example is from a postmortem meeting, where one thing that was mentioned was:

Incremental development: Partial deliveries are motivating. Externally visible.

This was later documented in the report as:

The idea of incremental development, i.e. partial deliveries to the customer, worked very well. The customer became more aware of the project status, and was able to guide and enforce changes at an early stage.

Which was again summed up as an experience item:

positive experience: partial deliveries

Later, we would then do categorizations based on these experience items.

3 Results

Here we first present how we conducted lightweight Postmortem Reviews. Then we describe the contents of each of the reports from the Experience Report and the lightweight Postmortem Review, and give some examples of the experience that was collected. We only give examples from one company to save space, but will use results from both companies when we go on to discuss the differences between the results of the methods in section 4.

3.1 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 [12] – 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 attached those as well.

Now, for the Postmortem Analysis meeting, we organized it with the following sections:

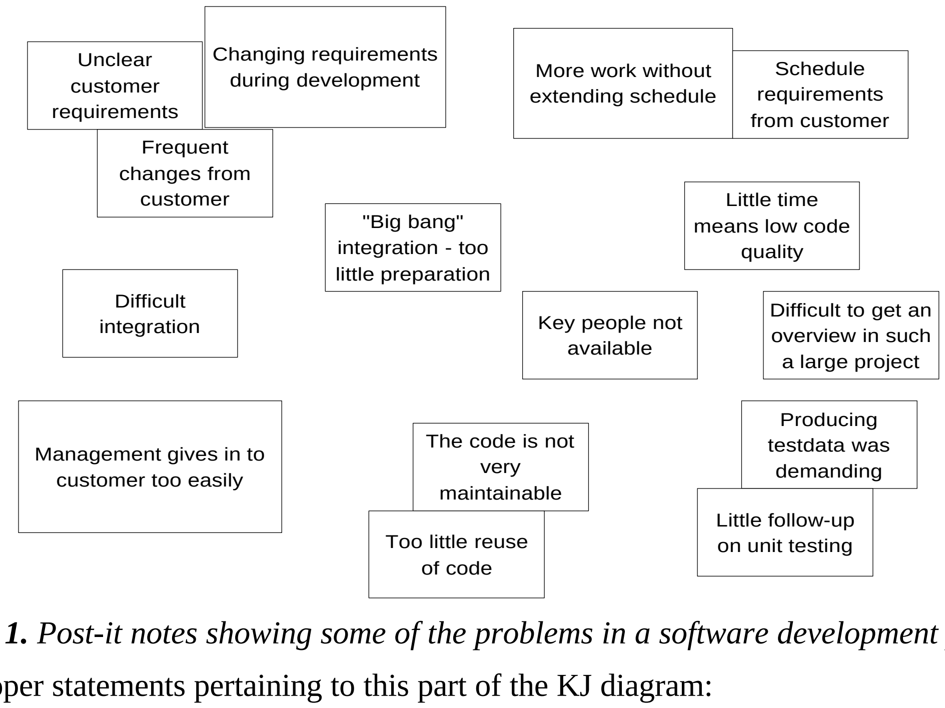
1. Introduction: First, we (the researchers) introduce the agenda of the day and the purpose of the Postmortem Review.
2. KJ session 1: We handed out post-it notes and asked people to write down what went well in the project, heard presentations, grouped the issues on the whiteboard, and gave them priorities.

3. KJ session 2: We handed out post-it notes and asked people to write down problems that appeared in the project, heard presentations, grouped the issues on the whiteboard, and gave them priorities.
4. Root Cause Analysis: We drew fish-bone diagrams for the main issues, the things that went well and the things that were problematic.

We used a tape recorder during the presentations, and transcribed everything that was said. The researchers wrote a Postmortem report about the project, which contained an introduction, a short description of the project that we analysed, how the analysis was carried out, and the results of the analysis. The result was a prioritised list of problems and successes in the project. We used statements from the meeting to present what was said about the issues with highest priority, together with a fishbone diagram to show their causes. In an appendix, we included everything that was written down on post-it notes during the KJ session, and a transcription of the presentation of the issues that were used on the post-it notes. In total, this report was about 15 pages long.
The day after the meeting, we presented the report to the people involved in the project to gather feedback and do minor corrections.
A comparison of this method to another way of performing lightweight Postmortem Review is discussed in another paper [13], where you also find information on the resources required.

3.2 Results from lightweight Postmortem Review

One result from the KJ session was two post-it notes grouped together and named "changing requirements". They are shown in the upper left corner of (some of) the results from the KJ process in Fig. 1.



Unclear customer requirements
Changing requirements during development
Frequent changes from customer
More work without extending schedule
Schedule requirements from customer
"Big bang" integration - too little preparation
Little time means low code quality
Difficult integration
Key people not available
Difficult to get an overview in such a large project
Management gives in to customer too easily
The code is not very maintainable
Too little reuse of code
Producing testdata was demanding
Little follow-up on unit testing

Fig. 1. *Post-it notes showing some of the problems in a software development project.*

Developer statements pertaining to this part of the KJ diagram:
"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."

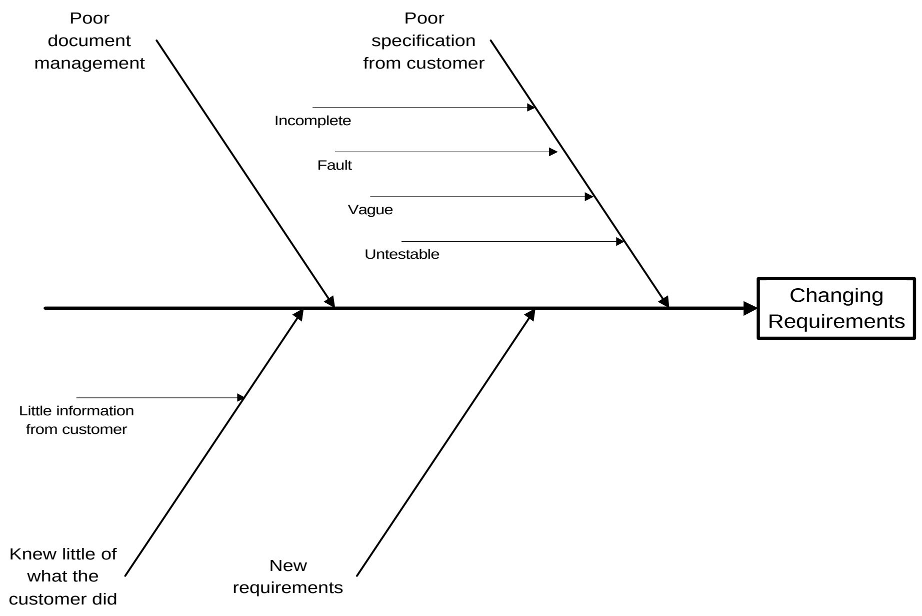

Poor document management
Poor specification from customer
Incomplete
Fault
Vague
Untestable
Changing Requirements
Little information from customer
Knew little of what the customer did
New requirements

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.
In total, we found 21 experience items using this method at Northern Software, where nine were "positive", and 12 were "negative". At Southern Software, we found 18 "positive" and 8 "negative" experience items, making it a total of 26.

3.3 Experience Report

The project manager wrote the experience report. Of nine pages of information, five was introduction and description of the project, and four pages contained descriptions of "problems during project execution" with proposals for improvement.
An example of a problem is the "Architectural Design Phase", which was described in the following way: "This phase should have been carried out in ten weeks according to the plan. We carried out this according to the plan, and was just two weeks late (this delay was introduced earlier in the project). We still got changes from (customer department) in this phase. The negotiations about the contract did not start before this phase was finished, and we worked for a while without an architecture proposal or a contract in a period. Much of the management work was done through the architecture design phase: Contract negotiation, revising schedules, etc".
To solve the problems that appeared during this phase, the project manager has the following suggestions for improvement: "In total, this phase worked out ok. We were 300 hours behind schedule after this phase, which mainly came as a result of clarifying

requirements from the customer. In the progress reports, I wrote that we should not give in to requests for changes in the estimates in the contract change notifications, or in the schedule. Experience indicated that we would get problems in the two next phases of the project, because the requirements were not stable, and the project was run according to (a standard process used in the company). I think we gave in to the customer in the negotiations about the schedule a bit easily, as well as the requirement on an intermediate delivery, which in a way worked as a template for the later negotiations. On the other hand, we got acceptance for the time estimates that we identified in contract change notification number 1 through 4."

At Northern Software, we found in total 28 experience items, where 27 were about problems, and just one about an activity that went well. The company produced this report with no inference from the researchers, who started working with them at a later stage. At Southern Software, we found 26 experience items, where 11 were "positive" and 15 "negative".

4 Discussion

Now, when we have seen a part of the reports from each of the two experience harvesting methods, can we say that there is any difference? Did the methods produce the same results? Could one method replace the other? To investigate this question, we studied the results in detail, and tried to group each of the experience items that were captured into one of the categories that we outlined in section 1.1.

We tried to apply the framework suggested by Novins and Armstrong, and to see if the experience could be said to be either *unique* or *programmable*, and *local* or *global*. Here, almost everything seemed to fit in the category *programmable* – the experience seemed to be pretty easy to transfer. To us, it looked like the experience would be valid for the whole company, that is, *global*. So with this framework, we could not distinguish between the results from the two methods.

We further found that almost all of the experience from both reports in both companies seemed to be of a kind that we can describe as "Problem understanding" [14, 15] – they were not as detailed as to use as a guideline, and we found little hard "facts" apart from in the introduction section in the Experience Report. Most of the information seemed to help "develop a better comprehension of a particular problem", and would be usable for "problem understanding".

Then, we tried to look at the topics that the experience was about. We made a list of *processes* in use in the companies, *actors* that would have different responsibilities, and also a category for *technology*, for experience related to tools, platforms and products. Using this type of categorization, and allowing one experience item to be related to both an actor, a processes or "technology", we found that the experience divided into the categories given in Fig. 3.

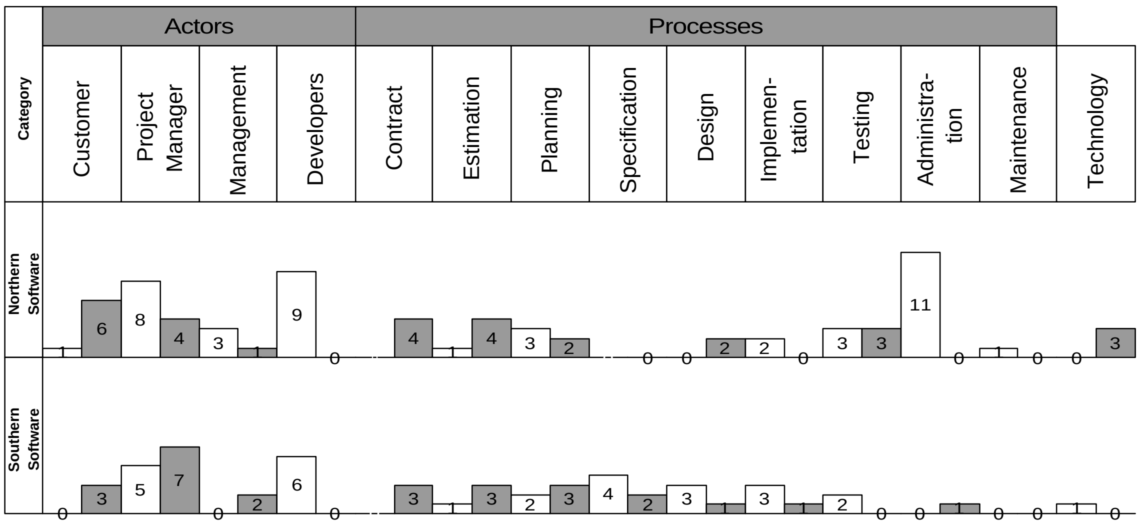


Fig. 3 *Experience according to categories. Results from the Lightweight Postmortem Review in white and from the Experience Report in grey.*

We see that most of the categories have experience from each of the methods, although the number varies. If we try to see which categories that have experience only from Postmortem Reviews, from both of the methods, and experience only
from the Experience Report, we find that the majority of experience categories are covered by both methods (6 categories). Postmortem Reviews covers one more (4) category than the Experience Reports (3) for Northern Software, but it is the other way around in the results from Southern Software. So it seems that the methods have covered slightly different issues.
To investigate this further, we examined the experience we had put in each category in more detail. Would these experience turn out to be the same, or different issues related to the same things in the same category? The results of this examination are given in Tab. 2. We have used the same set of categories as in Fig. 4, but replaced the name of each category with a number to save space. The number of categories that are "similar" means that "that number of the same experience items were found with both methods". The "total" figure refers to the sum of experience found with both methods.
From Tab. 2, we see that relatively few experience items are about the same issue. Only 5 experience items were found with both methods of a total of 69 experience items in Northern Software (some were put in several categories, so the sum of the "totals" in the table will be larger). At Southern Software, we found only three experience items with both methods, of a total of 49 experience items.

Tab. 2. *Similarities of experience from each of the categories. "Total" is the total number of experience items in each category. Results from postmortem reviews are abbreviated "postmortem", and experience report "experience".*

Category		*Actors*				*Processes*								
		1	2	3	4	5	6	7	8	9	10	11	12	13
Northern	Postmortem	1	8	3	9	0	1	3	0	0	2	11	1	0
	Experience	6	4	1	0	4	4	2	0	2	0	0	0	3
	Total	7	12	4	9	4	5	5	0	2	6	11	1	3
	Similar	*2*	*1*	*2*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
Southern	Postmortem	0	5	0	6	0	1	2	4	3	2	0	0	1
	Experience	3	7	2	0	3	3	3	2	1	0	1	0	0
	Total	3	12	2	6	3	4	5	6	4	2	1	0	1
	Similar	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*2*	*0*	*0*	*0*	*0*	*0*

So why is this? Why is it that we did not find a very large overlap in the results of the methods? One reason might simply be that by using a group process to elicit experience, we get to view what happened through several “different eyes”. A reason in the Southern Software case might be that the content of the lightweight Postmortem Review was known when the Experience Report was written. But it was the other way around in the Northern Software case.

Another thing we found, independently of the categorization frameworks, is that most of the experience from the Experience Report at Northern Software were about problems, whilst we intended to get 50% about problems and 50% about successes in the Postmortem Reviews. A reason for this might be that the Experience Report is used more to explain what went wrong than the Postmortem Reviews, which had a more precise goal of capturing experience that might be useful for others. At Southern Software, we did not find this in the Experience Report.

A difference we noted in Southern Software, was that the lightweight Postmortem Review would assign an experience to a situation and role, whereas the Experience Report would have a more managerial view of the experience, relating to the overall process. For example, enforcing formal change requests from the customer resulted in better cost control and planning from the management point of view. However, the developers considered that the main improvement was that they were relieved of direct, and frequent, customer interruptions.

5 Conclusions

Now we will sum up the conclusions we can draw from the discussion:

- The two methods find very little overlapping experience: We found more experience related to implementation, administration, developers and maintenance with the lightweight Postmortem Review. Whereas with the experience reports, we found more experience related to contract issues, design and technology.
- The experience items found with both methods seem to be usable for most projects within the companies, and seem to be quite easy to transfer.
- The two methods seem to find experience that can be used for problem understanding.

In all, it seems that both methods elicit information that are similar in style, but which is related to a bit different topics. If you are interested in issues related to the core processes of software development, lightweight Postmortem Reviews seems to be a better method than Experience Reports. If you are more interested in relations to the customer, Experience Reports seem to be a better choice.

Acknowledgements

We would like to acknowledge colleagues in the PROFIT project for providing a stimulating research environment, as well as our contact persons in Northern and Southern software. We are very grateful to Stefan Biffl at the University of Vienna and Reidar Conradi at the Norwegian University of Science and Technology for helpful comments on this paper. Furthermore, we are grateful to Geir Kjetil Hanssen at Sintef Informatics who managed part of the Lightweight Postmortem Review at Southern Software, and who made important contributions to the analysis. This work was supported by the Norwegian Research Council under project 137901/221.

References

[1] Karl M. Wiig, "Knowledge Management: Where Did It Come From and Where Will It Go?," *Expert Systems with Applications*, vol. 13, pp. 1-14, 1997.

[2] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach, "The Experience Factory," in *Encyclopedia of Software Engineering*, vol. 1, J. J. Marciniak, Ed.: John Wiley, 1994, pp. 469-476.

[3] Michael Polanyi, *The Tacit Dimension*, vol. 540. Garden City, New York: Doubleday, 1967.

[4] Peter Novins and Richard Armstrong, "Choosing your Spots for Knowledge Management," *Perspectives on Business Innovation*, vol. 1, pp. 45-54, 1998.

[5] Bonnie Collier, Tom DeMarco, and Peter Fearey, "A Defined Process For Project Post Mortem Review," *IEEE Software*, vol. 13, pp. 65-72, 1996.

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

[7] Watts S. Humphrey, "The Postmortem," in *Introduction to the Team Software Process*, *SEI Series in Software Engineering*. Reading, Massachusets: Addison Wesley Longman, 1999, pp. 185-196.

[8] Kurt Schneider, "LIDs: A Light-Weight Approach to Experience Elicitation and Reuse," presented at Second International Conference on Product Focused Software Process Improvement, PROFES 2000, Oulu, Finland, 2000.

[9] Reidar Conradi and Torgeir Dingsøyr, "Software experience bases: a consolidated evaluation and status report," presented at Second International Conference on Product Focused Software Process Improvement, PROFES 2000, Oulu, Finland, 2000.

[10] Davydd J. Greenwood and Morten Levin, *Introduction to Action Research*: Sage Publications, 1998.

[11] David Avison, Francis Lau, Michael Myers, and Peter Axel Nielsen, "Action Research," *Communications of the ACM*, vol. 42, pp. 94-97, 1999.

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

[13] Tor Stålhane, Torgeir Dingsøyr, Nils Brede Moe, and Geir Kjetil Hanssen, "Post Mortem - An Assessment of Two Approaches," presented at To be submitted to Achieving Quality in Software Engineering - AQUIS 2001, 2001.

[14] Chun Wei Choo, *The Knowing Organization : How Organizations Use Information to Construct Meaning, Create Knowledge, and Make Decisions*: Oxford University Press, 1998.

[15] R. S. Taylor, "Information Use Environments," presented at Progress in Communication Science, 1991.