A Case Study in Retrospective Analysis of Release Planning in an Agile Project

Lena Karlsson, Björn Regnell, Thomas Thelin

Abstract

This paper presents a case study evaluating a retrospective analysis method, aimed at improving the release planning activity in project management. The method is based on a re-evaluation of candidate requirements for prior releases in order to uncover release planning decisions that would have been made differently today. The method aims at using the lessons learned during the analysis to find improvement possibilities for the release planning in future projects. The release planning in the investigated project turned out as successful in the retrospective analysis. This may be due to the in-house project type, as the users in an in-house project are few and tangible, and have more similar requirements, compared to in a market-driven project. It may also be due to the iterative approach used during development as it provides possibilities of continual re-prioritisation of requirements.

1 Introduction

This paper presents a case study where a retrospective analysis method for the release planning activity was evaluated in an agile project. The method is called Post-release Analysis of Requirements SElection Quality (PARSEQ) and was first presented in [9]. The method aims at finding improvement suggestions for the release planning activity, as it is regarded as one of the most critical activities in market-driven software development [4]. Decisions regarding when, i.e. in which release, to include certain requirements may affect the success of the product.

In [9], the method was applied at a company developing a software product for an open market. The company had regular releases and used a requirements management (RM) tool [15] when planning their releases. Several improvement suggestions for the release planning activity was found in the case study, e.g. enhancing the overall picture of related requirements, increased attention to the elicitation of usability requirements and improved estimates of implementation costs.

This time we wanted to try the method out using a different approach. The project investigated in this case study had an agile development procedure, inspired by the Extreme Programming (XP) [1]. The goal of the project was to improve the IT support in the production system and was conducted as an in-house project, i.e. both users and developers of the project was from the same company. Since the project used an agile approach to development, they had frequent iterations and regular releases when the system was put into operation. For each iteration the project used the Planning game [1] to prioritise the requirements and plan the next release. The requirements were elicited in the beginning of the project from internal stakeholders and documented in an excel sheet.

The PARSEQ method is based on re-prioritising requirements from prior releases in order to find possible improvements for future releases. In the previous case study, the re-prioritisation was performed using the RM tool already available in the project. However, as the project described here, as many others, used an agile approach to development, we wanted to evaluate the application of the PARSEQ method in an agile project. Therefore, the Planning game was used to re-prioritise the requirements from prior releases. One main difference between the RM tool and the Planning game is that the RM tool provides the user with a priority list on a ratio scale, i.e. it is possible to see how much more important one requirement is than another,

tinually and release plans were flexible enough to adapt to the project scope. It may also be explained by the project type; users of in-house projects have similar requirements, and users and developers can co-operate during development as they are located close by. These are the two main issues that differ from the previous case, which investigated a project with a less agile development approach in a market-driven situation.

The paper is structured as follows. Section 2 describes some related work. Section 3 presents the method PARSEQ and its four steps. Section 4 provides background and description to the case under study. The results are discussed in Section 5 and the paper is concluded in Section 6.

2 Related work

This section describes the background of the methods used in this paper.

The agile methodologies have gained interest during the last years as it provides a more flexible alternative to the traditional development approaches. The most well-known approach may be Extreme Programming (XP) [1]. XP involves twelve practices such as Pair programming, Continuous integration, Refactoring, and the Planning game. The Planning game is a procedure used to determine the scope of the next release by combining business priorities, e.g. value to the user or customer, and technical estimates, e.g. time and resource estimates.

The retrospective is recognised as an important means for software process improvement [11]. The activity has many different names - process review, post mortem, project review, etc., but the idea is the same: by looking back and learning from the past we may find improvements for the future. The project retrospective is regarded as an excellent method for Knowledge Management [2]. One of the first papers to present a defined process recommends a five step approach that includes designing a project survey, collecting objective project metrics, conducting a debriefing meeting and a “project history day”, and finally publishing the results [5].

Nolan [13] states that the most effective way to improve is by *learning from success*. Rather than studying the failed projects or looking for external answers, you may choose to learn from the most successful projects. While traditional approaches to process improvement often focus strongly on improving through perpetual refinement, Nolan suggests that learning from success is a more effective and efficient scheme. He states: “If you believe that failure is no accident, then you must also believe that success is no accident”.

The release planning activity involves decisions regarding when and how to include different requirements in the product. It is considered as one of the most critical activities in software development [4]. Mistakes made during release planning may affect the success of the product. If a certain requirement is released too late, users may be annoyed if an expected function is missing. On the other hand, if a certain requirement is released too early, the foundation for it may be missing or the users might not be ready for it. The key is to use the available development resources in the best way possible, and put the functionality in the most appropriate release.

3 The PARSEQ Method

The basic idea of the PARSEQ method is the same as for other retrospective analyses, although in this case the issues regarding release planning and requirements prioritisation are selected for analysis and improvement. The goal of the method is to evaluate and find improvements for the release planning activity. The method is based on re-evaluation of a sample of requirements from prior releases. The re-evaluation is based on criteria such as implementation cost and user value and can be conducted using a requirements prioritisation technique to determine the current relative order among the requirements. During the time since the releases were launched new knowledge has been gained regarding the real implementation cost of requirements and the user response to the requirements. This knowledge is used to produce a post-release priority list with the relative order between the requirements in the analysis. In the root cause analysis, the release plan is compared to the post-release priority list in order to find requirements that are implemented either too early or too late. These requirements can point out how the organisation should have acted if they, earlier, had all the knowledge they have now. Thereby, it might be possible to improve the release planning process. The four steps in the method are illustrated in Figure 1.

The four steps described in this section can each be performed in different ways. In this case study we want to investigate how the PARSEQ method can be performed on a more agile project than has been investigated before. This is important since many projects do not store their requirements in a commercial RM tool, but rather use e.g. MS Excel™. Therefore, this case study investigates the PARSEQ method using the Planning game for re-estimation of cost and value in the second step. The Planning game is an agile approach to requirements prioritisation and can be used without any special tools. This section describes how the method was applied in the case study.

3.1 Pre-requisites for the PARSEQ Method

There are a number of practices that need to be implemented in order for the PARSEQ method to work properly:

- Multiple releases of the product and requirements from earlier releases are saved in a repository.

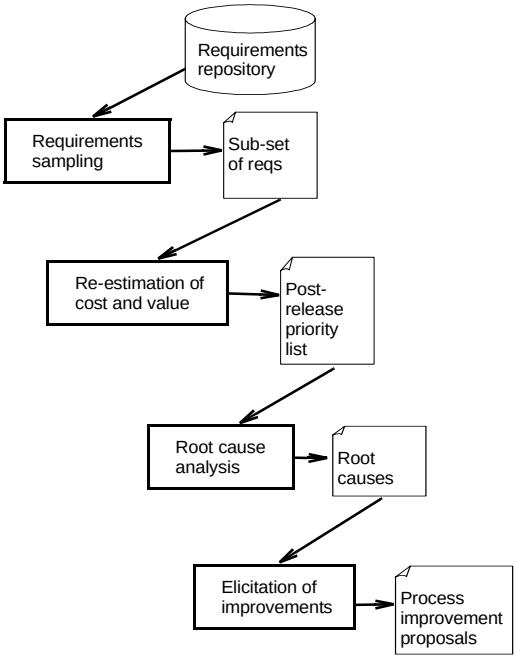

Fig. 1. An outline of the activities and products of the PARSEQ method.

- Data for each requirement stating which release it is implemented in, or if the requirement has been postponed or excluded.
- Employees who have decision-making experience from prior releases.
- A facilitator with experience of performing retrospective analyses.

The four steps in the PARSEQ method are described below.

3.2 Step 1: Requirements Sampling

The main input to the PARSEQ method is a requirements repository which comprises requirements that were candidates for the investigated product. Some of those requirements are implemented in one of the product releases and some are postponed or excluded.

The sample should be large enough to be representative for the product, but still small enough to be managed during one session. Each of the selected requirements should be mapped to the release it is implemented in – or if postponed, this should be noted.

3.3 Step 2: Re-estimation of Cost and Value

The requirements sample is used as input to the second step, where the cost and value of the requirements are re-estimated. Since these estimates may be difficult to retrieve in exact numbers, the second best approach is to re-prioritise the requirements and thereby get an ordered list of requirements. This does only give information about the relative order between the requirements, and not their value and cost in specific numbers.

The Planning game [1] is a prioritisation technique often used in agile approaches. It is based on dividing requirements into three groups, categorised e.g. high, medium and low. In order to get a complete priority order, the requirements are also ranked within the groups. To capture the common trade-off between user value and implementation cost, these are often used as criteria, although other criteria can be used as well.

We categorised the three user value groups as follows:

1. Requirements that are absolutely essential
2. Requirements that are less essential but still adds to the value
3. Requirements that are nice to have

Next the requirements are prioritised based on implementation cost. We categorised the three implementation cost groups as follows:

1. Essentially more than medium cost
2. Medium cost
3. Essentially less than medium cost

Preferably, the value criterion should be re-estimated by key users of the product or by marketing personnel who have a good view of the customer wishes, and the implementation cost should be re-estimated by developers who are involved in the product development.

When the requirements are grouped and ranked within the groups, a *tail-head comparison* can be performed to ensure the correct order is obtained [8]. This is performed by comparing the lowest ranked requirement in one group with the highest ranked requirement in the next group. The comparison is continued until all requirements are in the correct order.

The cost/value approach to release planning has been shown useful to illustrate the trade-off decision makers often face [7]. In order to produce a foundation for the root cause analysis, a cost/value diagram is constructed based on the ranked requirements.

3.4 Step 3: Root Cause Analysis

The root cause analysis consists of a discussion about the different fields in the cost/value diagram. Ideally, the requirements to the upper left (area I in Figure 2) should be implemented first, since they have a high-value and low-cost combination. The requirements to the lower right (area III in Figure 2) should be implemented last or not at all,

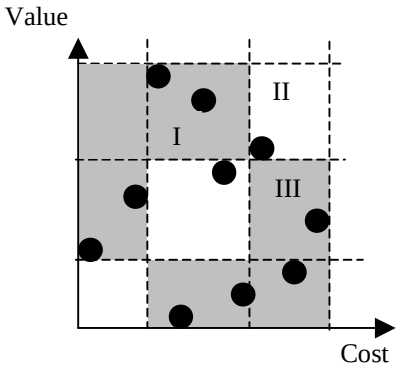

Fig. 2. Cost/value diagram example with the areas to investigate

since they have a low-value and high-cost combination. Therefore, the requirements in these areas are subject to investigation as we want to find the requirements that were implemented either too early or too late.

Some questions can be used as guidance to find root causes for different decisions:

- Why did we implement the requirement that early/ late?
- Based on what facts, did we make that decision?
- What has changed since the decision was made?
- Was it a correct or incorrect decision?

This results in a mapping between the root causes and the requirements that are based on incorrect decisions.

3.5 Step 4: Elicitation of Improvements

The outcome of the root cause analysis is used to facilitate the elicitation of improvements. The intention is to base the discussion on strengths and weaknesses of the requirements selection process and identify changes to current practices. A number of questions can assist to keep focus on improvement possibilities:

- How could we improve decision-making?
- What would be needed to make a better decision?
- Which changes to the current practices can be made to improve requirements selection in the future?

The results can be used in a process improvement programme aiming at enhancing the decision-making in future releases.

4 PARSEQ Case Study

This section describes the case study in which the PARSEQ method was evaluated in an agile setting. The case study involves an in-house project using an agile development approach. The case study was performed as a part of a research collaboration between the researchers and the participating company. Below the case study background is described, along with the results from the case study execution. The section ends with the threats to validity.

4.1 Case Study Background

The organisation that participated in the study develops embedded software products for a global market. The case in focus is an in-house project aimed at improving the production IT system, and its connection to the business system and production database. The production IT system is divided in three main steps. First, items are provided with software and tested, then items are labelled and placed in a suitable box, and finally items are bundled into multipacks that should be shipped to a certain customer.

The project used an agile development method, inspired by Extreme Programming [1], with frequent releases. Although the project had never tried working in an agile manner before, some co-workers had experience in agile methodologies.

A repository of requirements was developed early on in the project, by interviewing a large number of stakeholder representatives and documenting their wishes for the new system. One of the researchers was involved as assistant in the elicitation of requirements for the system. The project was divided into several releases of the product, and each release was divided into several shorter iterations. In the beginning of each iteration, a Planning game activity was performed in order to determine which requirements to implement. The requirements repository was used as input to the planning and prioritisation, although the requirements had to be broken down to a more detailed level when planning the iterations.

4.2 Results from Executing the Case Study

The case study was divided into two separate occasions since the users were unable to attend the first session. The Project manager and the System architect participated during the first session, as they were involved throughout the development of the system. The second session was attended by the Production manager and the Production test manager who are key users of the production system. They also acted as customers of the project and suggested many of the requirements in the repository.

One of the researchers acted as facilitator and took notes during both occasions. The Requirements sampling step took about one hour and was performed by the facilitator before the first session. The sessions on site lasted for approximately two hours each.

Step 1: Requirements sampling

The requirements repository consisted of about 120 requirements arranged by the date it was entered. In order to get a reasonable sample, every fourth requirement was selected, i.e. the sample consisted of 30 requirements. Thus, the sample included requirements suggested all through the elicitation phase. The sample included both requirements that were implemented during the project and requirements that were postponed or excluded.

As we intended to use the Planning game as re-estimation technique in step 2, the requirements were printed on two sets of cards, one representing the user value and one representing the implementation cost.

Step 2: Re-estimation of value and cost

The requirements sample was re-estimated based on cost and value. The intention was to let the Project manager and System architect estimate the implementation cost, as they represent the developers of the system, and the Production managers estimate the user value, as they represent the users of the system. However, as the users were unable to attend, the developers had to play both roles during the first session.

The participants were instructed to start with the value criterion, representing the value to the users. They were asked to create three rather evenly large groups, i.e. none of the groups should be almost empty. The three groups were divided as shown in Table 1. The developers only prioritised 29 of 30 requirements, since they classified one requirement as impossible to prioritize.

The participants were then asked to rank the cards within each group, in order to get a list of prioritised cards. Without instructions, they used a sorting technique and took one card and compared it to the others in the list to see where to insert it. They spent a bit more time prioritising on the cost criterion than on the value criterion, see Table 2. This may indicate that the cost criterion was more difficult for them to estimate than the user value.

In the second session, the users prioritised the requirements based on both value and cost for the purpose of comparing their priorities to the developers'. As can be suspected, the implementation cost was difficult for them to estimate. The users found it easier than expected to prioritise based on value, but as they did not have any clue about implementation costs, they used their best guess. They spent less time on the cost criterion than on the value criterion. The users applied another approach when the three groups had been formed, and divided each group into three new groups until they had reached a complete ranking.

Table 1. Number of requirements in each priority group

Value groups	Developers	Users	Cost groups	Developers	Users
Reqs that are absolutely essential	16	15	Essentially more than medium	9	9
Reqs that provides added value	6	8	Medium	11	10
Reqs that are nice to have	7	7	Essentially less than medium	9	11

Table 2. Time spent on prioritisation

Time (minutes)		Developers	Users
Value	Divide into groups	15	13
	Within groups	15	11
Cost	Divide into groups	20	7
	Within groups	20	9

As the users worked rather fast, they were given a chance to use tail-head comparison. The lowest ranked requirements in the top value group was exchanged for the highest ranked requirements in the next group before they were satisfied. During prioritisation of implementation cost, they did not change any of the requirements in the tail-head comparison.

Analysis of agreement. The Kappa value (K) can be used to assess the agreement between a set of raters who assign a set of objects into a set of categories [14]. In this case, the three fields (I, II, III) in Figure 2 were used as categories into which objects, i.e. requirements, were assigned. The two sets of raters were the users and the developers. The Kappa values are presented in Table 3.

Table 3. Kappa value for a comparison between users' and developers' estimates of cost and value

Criteria	Kappa value
Cost	-0.08
Value	0.32

Different suggestions have been presented regarding the interpretation of the Kappa value. In [6], one suggestion is that $0.21<K<0.40$ can be regarded as a *fair agreement*, which would be the case for the agreement between the users' and the developers' estimates of value. Kappa values close to, or below, zero suggest *no agreement*, and therefore there would be no agreement between the users' and developers' estimates of cost.

Cost/value approach. As the cost estimates made by the users were too unreliable, we have chosen not to include their cost/value diagram here. The value estimates made by the developers turned out to be rather similar to the value estimates made by the users, as the Kappa value indicates.

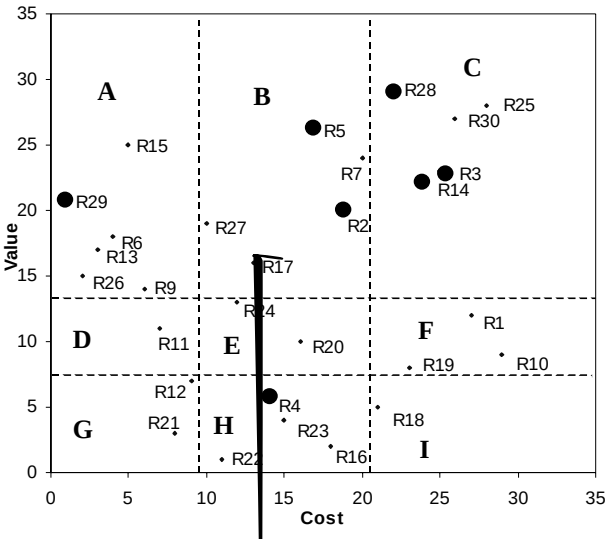

Figure 3 shows the cost/value diagram performed by the developers.

The facilitator drew a diagram on the whiteboard, with the y-axis representing the ranked order of user value and the x-axis representing the ranked order of implementation cost. The value cards were used to place the requirements in the correct order based on value, i.e. vertically on the cost/value diagram. The cost cards were then used to place them in the correct order based on implementation cost, i.e. horizontally. The facilitator also drew lines representing the three groups, resulting in nine sections (A-I) on the whiteboard.

Step 3: Root cause analysis

The root cause analysis consisted of a discussion about different fields in the cost/value diagram. The project included two releases, which consisted of several iterations. Sections A, B and D, ideally, ought to be implemented from the first release and sections F, H and I should be implemented in the third release or not at all. They discussed the requirements in sections A, B and D and the requirements that were not implemented in this release.

The cost/value diagram made by the developers, with the requirements that appeared in sections A, B and D implemented in the system, although four requirements included in the first release. The discussion about why the requirements could not be implemented in this release as other requirements were needed as foundation. They were not needed urgently, but they were very important to include *sometime*. Therefore they could be postponed to a later release when it was better suited to implement them. Only one of the twelve requirements was excluded. However, the developers stated that the basic problem behind the requirement was solved in another way.

Sections F, H and I contained eight requirements of which seven had not been implemented. One of the requirements had been implemented in the first release. It was a usability requirement regarding the user interface. It was expected to have a higher user value during development than it actually had when the system was put into operation. Therefore, it was implemented although, with regard to today's knowledge, it should probably not have been included in the system.

The root cause analysis performed in the second session, i.e. with the users, was difficult to draw any conclusions from. The cost/value diagram pointed out some requirements to investigate, but since there were no developers around to answer questions about why, and if, requirements were implemented in a certain release, it was difficult to find any root causes. It was also difficult to trust the indications from the cost/value diagram, since the users knew their cost estimates were unreliable. Therefore it was decided to only use the users' estimates as comparison with the developers' estimates, which was presented earlier.

Step 4: Elicitation of improvements

The PARSEQ analysis indicated that the release planning in this project was successful. The developers concluded that one reason for the successful release planning was the iterative development. During the project, the Planning game was used to evaluate and prioritise the requirements regularly, and the release plan was flexible enough to adapt to changes in the requirement priorities and in the project resources.

Thus, the most important insight was that regular prioritisation yields better release plans and this lesson will be brought into other projects at the company. More prototyping activities during release planning was also mentioned as a possible improvement in other projects. The participants were pleased with the result, as it was a confirmation that they had prioritised and decided correctly and that the iterative development is a successful way of working.

Although the users' root cause analysis did not bring any particular conclusions, the users were surprised with the extent of which they were actually given the most important functionality. Since the scope of the project was cut down during development due to lack of resources, the users were worried that some important functionality had been excluded. The retrospective analysis showed that this was not the case, which of course is encouraging for all stakeholders.

4.3 Threats to Validity

The case study was the first time the PARSEQ method was used with the Planning game as re-prioritisation technique. It was also the first time it was used on an in-house project using an iterative development approach. The most important threats to validity are described below.

Ideally, the cost/value diagram should be designed using developers' cost estimates and users' value estimates and the retrospective session should be attended by both roles. As the analysis was divided in two sessions in the case study, it may have affected the results. The developers estimates of value agreed rather well with the users estimates, but if the users' estimates had been available, a slightly different set of requirements would possibly have been pointed out in the root cause analysis. If both users and developers had been present it could also have brought a more interesting discussion between the parties.

The sample was selected from a list of requirements arranged by date of arrival. Therefore, no consideration was made to which release the requirements belong to. Only a few were implemented in the second and third releases, while many were implemented already in the first release. Requirements more evenly distributed between the releases could have yielded a more interesting analysis. Similarly, the analysis could have been improved if not as many requirements were postponed.

The requirements repository that was used as input to the study contained high-level user requirements and therefore it was sometimes difficult for the participants to judge whether or not a requirement had been implemented – some were implemented partially and some were implemented over a large period of time. Therefore, the mapping between requirements and release number is approximated. It would possibly have been easier to do the mapping if more detailed requirements were used instead of user requirements. However, it is desired to get the users' view of the system, which may be difficult if analysing too detailed requirements.

Some issues regard the method rather than the case studied here, and need further attention in future case studies. Several of the requirements in the analysis were subject to interdependencies. Some requirements affected the system architecture and had to be implemented before others. This complicated the re-evaluation of value as well as cost. The developers tended to give the more fundamental requirements a higher priority, as they needed to be implemented early on. Thus, the value to the users had to stand back. This may be one explanation to the difference between users' and developers estimates of value. Another explanation could be that the cost of a specific requirement is actually dependent on which release the requirements are implemented. This dependency is difficult to evaluate for a user of the system.

Requirements prioritisation is based on subjective judgements, and may vary between different stakeholders. Therefore, another set of participants could have produced a different cost/value diagram and thereby pointed out different requirements to analyse. The threat was reduced by letting two user representatives and two developer representatives co-operate and negotiate on the priorities.

5 Discussion

In a prior case study the PARSEQ method was evaluated in a software developing company [9]. The retrospective analysis focused on a commercial software product that is sold on an open market. The findings included several improvements to the release planning process found during the root cause analysis. The company discovered that they needed to enhance both the overall picture of related requirements and the division of large requirements into smaller increments. They also found out that usability requirements needed more attention in the elicitation phase. The company also tended to estimate the market-value of features in competing products too high, while effort estimates were found to be both too high and too low. The participants found the exercise interesting and instructive.

The results from the PARSEQ analysis in this case study indicate that the release planning in the investigated project have been successful. Two main reasons for the successful release planning have been discussed. First of all, as suggested by the developers, the iterative development and continual re-prioritisation provided a flexible release plan that could be revised and adapted to changes in the requirement priorities and in the project resources. Prototyping activities was also mentioned to have improved release planning as user feedback could be taken into consideration.

A second possible reason is the type of project that was investigated. In the previous case study, a market-driven product was investigated in the retrospective, while in this case study an in-house project was put under investigation. The users in an in-house project are few and more tangible, and have more similar requirements for the system. Users of a commercial product can use the system in many different ways, sometimes in ways unknown to the developer. The users' requirements are therefore more scattered and diverse for a market-driven product. This may be one of the reasons for release planning appearing successful – the users' needs were easier to find early on and the developers understood the users' opinions as they are all within the same company.

One goal of this case study was to investigate the PARSEQ method in an agile project, in comparison to the

previous case study. Since most projects do not have requirements stored in a commercial tool, it is interesting to investigate how a manual prioritisation technique works for the second step of the PARSEQ method. According to the participants it was easy to use the Planning game procedure to rank the requirements. However, the Planning game only presents the requirements on an ordered list, while the RM tool, used in the previous case study, also presents the ratio between requirements priorities. This difference may affect the results so that another set of requirements is pointed out in the cost/value diagram. This is because in this case the cost/value diagram is based on ranks, while in the previous case it is based on ratios, so some requirements might end up in another root cause area (I, II, or III). However, it seems as the Planning game may be sufficient for our purpose.

6 Lessons learned and future work

This section discusses the lessons learned during the case study and future research of PARSEQ. As was shown in a previous case study, several improvement suggestions were found for the market-driven project [9]. A conclusion is that the PARSEQ method is probably more valuable to market-driven projects as they may have a more problematic situation during release planning.

The main lessons learned from this case study are:

- It is necessary that developers and users attend the same session in order to gain as much as possible from the root cause analysis.
- Most of the requirements that were investigated in the root cause analysis were actually not implemented in an incorrect release. This since there were requirements dependencies that forced some requirements to be implemented before others. This is important in release planning and it is discovered in the retrospective analysis. Thus, not all requirements pointed out in the cost/value diagram are based on incorrect decisions.
- Prototyping and iterative development work well in software development. These are examples of *learning from success* as described in the introduction [13]. Evidently, the release planning had been a success in the described case, and the company can learn from that. The participants concluded that prototyping is a good way to validate the requirements, and that the iterative development approach provided the developers with the most acute requirements for each iteration.

Future research includes three main things:

- Investigate possible tool support for the PARSEQ method as it could speed up the method and leave more room for discussions. It could provide us with different possibilities to perform the different steps in the method and guide the facilitator during the process.
- As it has been shown that the Planning game is just as accurate at ranking requirements as the techniques based on pair-wise comparisons [10], it would be interesting to investigate the difference between techniques using an ordinal scale and techniques using a ratio scale. We could investigate if the PARSEQ method needs the ratio scale, or if ordinal scale is sufficient. We also need to evaluate how the areas in the cost/value diagram should be selected in order to point out the most interesting requirements to investigate.
- There may be other criteria than cost and value that determines how the releases should be planned, as many different criteria affect the priority in practice. Some other criteria are described in [12], e.g. effort, resources, and logical implementation order, which could be used instead of cost. Further, importance to key customers, importance to users, product strategy, and company profit could be used instead of value. This may be investigated further as the most appropriate criteria may vary depending on situation, product or company investigated.
- It would be interesting to investigate another case where iterative development was used in order to see if the conclusion drawn here can be corroborated, i.e. that iterative development with frequent iterations can lead to more successful release planning.

Acknowledgements:

The authors would like to thank the four participants in the case study, who have contributed with their time and experience and participated in valuable discussions regarding requirements engineering, release planning, and possibilities for the PARSEQ method.

References

[1] Beck, K., *Extreme Programming Explained*, Addison-Wesley, 1999.

[2] Birk, A., Dingsøyr, T., Stålhane, T., “Postmortem: Never Leave a Project Without It”, *IEEE Software*, pp. 43-45, May/June, 2002.

[3] Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J., “An Industrial Survey of Requirements Dependencies in Software Product Release Planning”, *5th Int. Symp. on Requirements Engineering*, pp. 84-91, Toronto, Canada, 2001.

[4] Carlshamre, P., “Release Planning in Market-Driven Software Product Development: Provoking an Understanding“, *Requirements Engineering*, Vol. 7, pp. 139-151, 2002.

[5] Collier, B., DeMarco, T., Feary, P., "A Defined Process for Project Postmortem Review", *IEEE Software*, pp. 65-72, July, 1996.

[6] El Emam, K., "Benchmarking Kappa: Interrater Agreement in Software Process Assessments", *Empirical Software Engineering*, Vol 4, pp. 113-133, 1999.

[7] Karlsson, J., Ryan, K., "A Cost-Value Approach for Prioritizing Requirements", *IEEE Software*, pp. 67-74, Sept/Oct 1997.

[8] Karlsson, J., Wohlin, C., Regnell, B., "An Evaluation of Methods for Prioritizing Software Requirements", *Information and Software Technology*, Vol. 39, pp. 939-947, 1998.

[9] Karlsson, L., Regnell, B., Karlsson, J., Olsson, S., "Post-Release Analysis of Requirements Selection Quality - An Industrial Case Study", *9th Int. Workshop on Requirements Engineering: Foundation for Software Quality*, Velden, Austria, 2003.

[10] Karlsson, L. Berander, P., Regnell, B., Wohlin, C., "Requirements Prioritisation: An Experiment on Exhaustive Pair-Wise Comparisons versus Planning Game Partitioning", *Proceedings of the 8th Int Conference on Empirical Assessment in Software Engineering (EASE'04)*, Edinburgh, Scotland, UK, 2004.

[11] Kerth, N.L., *Project Retrospectives: A Handbook for Team Reviews*, Dorset House Publishing, 2001.

[12] Lehtola, L., Kauppinen, M., Kujala, S., "Requirements Prioritisation Challenges in Practice", *Proceedings of the 5th Int. Conference on Product Focused Software Process Improvement*, pp. 497-508, Japan, 2004.

[13] Nolan, A.J., "Learning from Success", *IEEE Software*, pp. 97-105, Jan/Feb, 1999.

[14] Siegel, S., Castellan, J.N., *Nonparametric Statistics for the Behavioral Sciences*, Second Edition, McGraw-Hill, 2000.

[15] http://www.focalpointus.com (visited June 2005)