Lehtinen, Timo O A; Itkonen, Juha; Lassenius, Casper

Recurring opinions or productive improvements—what agile teams actually discuss in retrospectives

Published in:
Empirical Software Engineering

DOI:
10.1007/s10664-016-9464-2

Published: 01/10/2017

Document Version
Publisher's PDF, also known as Version of record

Empir Software Eng (2017) 22:2409–2452
DOI 10.1007/s10664-016-9464-2

Recurring opinions or productive improvements—what agile teams actually discuss in retrospectives

Timo O. A. Lehtinen[1] · **Juha Itkonen**[1] · **Casper Lassenius**[1]

Published online: 3 November 2016

Abstract Team-level retrospectives are widely used in agile and lean software development, yet little is known about what is actually discussed during retrospectives or their outcomes. In this paper, we synthesise the outcomes of sprint retrospectives in a large, distributed, agile software development organisation. This longitudinal case study analyses data from 37 team-level retrospectives for almost 3 years. We report the outcomes of the retrospectives, their perceived importance for process improvement and relatVed action proposals. Most discussions were related to topics close to and controllable by the team. However, the discussions might suffer from participant bias, and in cases where they are not supported by hard evidence, they might not reflect reality, but rather the sometimes strong opinions of the participants. Some discussions were related to topics that could not be resolved at the team level due to their complexity. Certain topics recurred over a long period of time, either reflecting issues that can and have been solved previously, but that recur naturally as development proceeds, or reflecting waste since they cannot be resolved or improved on by the team due to a lack of controllability or their complexity. For example, the discussion on estimation accuracy did not reflect the true situation and improving the estimates was complicated. On the other hand, discussions on the high number of known bugs recurred despite effective improvements as development proceeded.

Communicated by: Magne Jørgensen, Mika Mäntylä, Paul Ralph and Hakan Erdogmus

✉ Timo O. A. Lehtinen
timolehtinen@iki.fi

Juha Itkonen
juha.itkonen@aalto.fi

Casper Lassenius
casper.lassenius@aalto.fi

[1] Department of Computer Science, Aalto University School of Science, P.O. BOX 15400, FI-00076 Aalto, Finland

Keywords Retrospective · Software engineering · Scrum · Agile · Continuous improvement · Longitudinal case study

1 Introduction

Retrospectives, also known as 'post-mortems' and 'post-project reviews' have often been proposed as a tool for making improvements in software development activities due to their theoretical capability to generate learning and recognise the success and failure of software engineering practice (Dingsøyr 2005).

Modern software development methods, such as Scrum (Schwaber and Sutherland 2011), have emphasised the role of retrospectives during the development period as opposed to after it has ended in helping enable continuous learning and process improvement. In agile development, retrospectives are recommended to be applied continuously; for example, sprint retrospectives can be applied after each sprint review when using the Scrum method (Schwaber and Sutherland 2011), and some authors suggest spending 1 h every other week reflecting on working habits (Cockburn 2002). Since applying retrospectives in this way requires considerable effort from the software teams, it is important to understand the results of such a practice in order to assess its value.

Despite this need for and the widespread use of retrospectives, their tangible outcomes have received little research attention. Prior studies have certainly noted the need for retrospective and introduced various methods for conducting them (see section 2 for details). However, what actually is discussed and decided in retrospectives — negative and positive observations as well as corrective actions — have not been reported, synthesised or compared in previous studies. More importantly, prior studies have not analysed how the outcomes of the retrospectives change over time. Such a longitudinal perspective can help us understand the extent to which the reported observations and corrective actions keep repeating themselves and how the topics and improvement ideas change over time. This will help experts evaluate the role of continuous retrospectives for process improvement and knowledge elicitation.

In this article, we start to fill in this gap in the existing literature by analysing the outcome of team-level retrospectives in a longitudinal case study of a large, distributed Scrum organization. We analyse the discussion topics and their evolution over time as well as the developed corrective actions. The data consist of the retrospective diagrams recorded by the participants in the team-level retrospective meetings. The raw statements extracted from the retrospective diagrams are categorised based on the process areas and topics. Our analysis is based on these categorisations and a qualitative analysis of the contents of the retrospective statements.

The rest of the paper is organised in the following way. Section 2 introduces related work regarding different types of retrospective methods and the motivators for conducting analyses on positive and negative software development experiences. Section 3 presents the research questions and the case study design, including the retrospective techniques used in the case organisation. Section 4 presents the case study results together with the discussion based on the research questions. Section 5 provides a discussion on the implications the research findings and presents a validity evaluation of the research. Section 6 concludes the study and discusses opportunities for future research.

2 Related Work

2.1 Methods for Conducting Retrospectives in Software Engineering

Retrospective methods have been defined as 'collective learning activities' (Dingsøyr 2005), which are used to recall software development experiences and problems in order to make improvements to future software development practice. Generally speaking, retrospectives aim to recognise software engineering successes and failures and to link the related experience to action proposals that will improve future successes and decrease the likelihood of future failures. The tangible outcome of a retrospective is a report (Dingsøyr 2005), which may include positive and negative experiences (Bjørnson et al. 2009), detected problems (Lehtinen et al. 2011; Stålhane et al. 2003), voting results (Lehtinen et al. 2011) and action proposals (Lehtinen et al. 2011). Retrospectives are used at various levels, including the level of software teams, organisations and companies (Lehtinen 2014). They can be mapped onto development phases, eg in the form of 'planning post-mortems', 'design/verification post-mortems' and 'field post-mortems' (Tiedeman 1990).

Agile software development methods emphasise the importance of using retrospectives continuously according to agile principles: 'At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly' (Cockburn 2002). Such retrospectives have to be constructive and lightweight in terms of the effort expended and the required calendar time (Glass 2002). In agile retrospectives, it is common to reflect on the experiences of the past development iteration within the team, identify what has worked well and what are the problems and their causes, and then to make decisions on what improvements to make (Derby et al. 2006; Cockburn 2002). A typical agile retrospective is the sprint retrospective in Scrum, where the Scrum team has a roughly three-hour, time-boxed retrospective meeting after each development sprint (Schwaber and Sutherland 2011). Agile retrospectives in general can adopt a variety of techniques and tools for analysing the development successes and challenges, such as the five whys, fishbone diagrams and voting techniques (Derby et al. 2006).

Retrospective methods often include three steps: 1) target definition, 2) reflection (sometimes referred to as 'root cause analysis') and 3) corrective action development. These steps have been discussed in detail in a study by (Lehtinen 2014). The *target definition* aims to focus the scope of the retrospective analysis on a relevant and important topic. Such topics have included the causes of software defects (Jalote and Agrawal 2005; Card 1998; Lehtinen et al. 2014a), the detection of negative and positive development experiences (Schwaber and Sutherland 2011; Bjørnson et al. 2009), analysis of the requirements problems (Tiedeman 1990; Jin et al. 2007), lead-time problems (Lehtinen et al. 2014a), installation problems (Lehtinen et al. 2014a) and expectation mismatches (Lehtinen et al. 2015b).

Reflection is the step where negative and positive experiences are identified and analysed. At this stage, methods such as *root cause analysis* can be used (Collier et al. 1996; Derby et al. 2006; Dingsøyr 2005; Card 1998). Reflection makes it possible to structurally investigate a problem and recognise the underlying causes needed to prevent it from occurring in the future. With retrospectives, root cause analysis has been applied with lightweight diagramming techniques using a team investigation process that aims to understand why and how the target problem occurred. Root cause analysis has also been used to explore the factors of positive software development experiences. While the existing literature discusses these methods, little has been said about what they actually produce, how well that matches with reality and if the

problems are actually being solved or if they simply recur over and over again when conducting continuous retrospectives.

Corrective action development is the third step used in retrospective methods. This step can be initialised by having the retrospective participants vote on the identified causes (Lehtinen et al. 2014a). Voting helps steer the process of developing corrective actions in the direction of defining improvement targets that the participants perceive as being worthwhile to achieve. Corrective action development techniques are the least studied area in the context of software engineering retrospectives (Lehtinen 2014). Researchers have suggested that such techniques should also include brainstorming and brainwriting exercises as well as the use of interviews after the retrospective meetings as feasible approaches for developing corrective actions.

2.2 The Outcomes of Lightweight Software Engineering Retrospectives

To the best of our knowledge, the tangible outcomes of software engineering retrospectives have been rarely studied. The main gaps in the existing literature include an understanding of the continuous retrospective practice, what the outcomes are and how the outcomes change over the time. Additionally, there is lack of knowledge about retrospective outcomes that focus on positive experiences. None of the studies we identified recognised the links between the different types of discussion topics and the development of corrective actions, which would be valuable for steering the discussions into constructive process improvement topics.

Dingsøyr et al. (2001) has presented the outcome of lightweight retrospectives in the form of sample outcomes from two software companies. The outcomes indicate problems in various process areas (contract, estimation, planning, etc.) and related to various actors (customer, project manager, developer, etc.). Stålhane (2004) has likewise presented the outcomes from two retrospective approaches in a software company. His results also include problems encountered in various process areas, including delivery, development, management and organisation. Similarly, Lehtinen et al. (2014a) focused on retrospectives in four medium-sized software companies. Their findings are in line with those of Dingsøyr et al. (2001) and Stålhane (2004), indicating that retrospectives can reveal problems from various software development process areas (sales & requirements, management, implementation work, etc.) and with respect to various types (cooperation, existing product, resources & schedules, etc.). Furthermore, Lehtinen et al. (2014a) indicate that retrospectives can reveal perceived causal mechanisms between the detected problems interconnecting the retrospective outcomes. The outcomes from the two retrospectives introduced by Stålhane et al. (2003) support this hypothesis.

There is a gap in the existing literature regarding retrospective outcomes, especially over time. A detailed analysis of the topics of discussion and corrective actions as the outcome of retrospectives has not been performed. There is a lack of understanding of the interrelationships between the discussion topics and the corrective actions, what topics are discussed a great deal and what topics best lend themselves to corrective actions. Furthermore, there is a need to study how the retrospective outcomes change over time; the team-level retrospectives should be applied as a continuous practice in agile software development. It is beneficial to study whether or not continuous retrospectives remain productive over time, as well as whether the same discussions keep recurring over time. To the best of our knowledge, no existing study has covered the outcome of continuous retrospective practice or analysed how the retrospective outcomes change over time, if at all.

Studies on retrospective outcomes with respect to positive software development experiences are largely missing. Bjørnson et al. (2009) presented the outcomes of team-level retrospectives via two examples. However, they did not thoroughly analyse the outcomes. Nonetheless, the study illustrates the retrospective outcomes of positive experiences, which other studies have not done.

In a previous study, we developed and evaluated the root cause analysis methods used in software project retrospectives (Lehtinen et al. 2011). We applied the methods to an analysis of the causal mechanisms leading to failures or critical problems in software projects and product development (Lehtinen et al. 2014a, 2015b). We also developed a supporting tool to assist in recording the retrospective outcomes (Lehtinen et al. 2014b). In these studies, the root cause analysis method was applied in the context of focused organisation-level retrospectives that analysed a specific critical problem. Our results indicated that the use of root cause analysis can improve the retrospective outcomes by providing in-depth analysis for important software engineering problems, and we suggested that root cause analysis was perceived as a feasible approach for conducting software project retrospectives. In our earlier studies, the retrospectives were organised as a single event consisting of one or more retrospectives facilitated by the researchers. In this article, we study a continuous, team-level retrospective practice using a longitudinal approach by analysing retrospective outcomes from several teams over a three-year period. One of the case organisations from our earlier work (Lehtinen et al. 2015b) is the same as the case in this study, but the dataset in this study is different. The software tool that was developed in our earlier work was also used in the retrospectives by the teams working in the case organisation in this study.

3 Case Study Design

This paper is based on a longitudinal case study (Yin 1994) in a single case organisation. The study is longitudinal (Runeson et al. 2012) because observations of the studied phenomenon were collected over a long period of time and, in addition, the researchers had a prolonged involvement in the case company through research collaboration over multiple years. A longitudinal case study is an appropriate means to study research questions concerned with the outcomes of team-level retrospectives when applied as a continuous practice in every development sprint over a long period of time.

In this section, we first present the research objective and the detailed research questions. After that, we introduce the case study context, the case organisation and the retrospective methods applied in the case organisation. Finally, we describe the data collection process and analysis methods.

3.1 Research Objective and Questions

Our objective was to create descriptive knowledge about the outcomes of team-level sprint retrospectives by synthesising and analysing a longitudinal dataset on the retrospectives done in a single software development organisation. Our goal was both to describe and categorise the outcomes of the retrospectives and to analyse how they evolve over time when applied as a continuous practice.

We refer to the recorded retrospective statements, both positive and negative, and the developed corrective actions as the *outcomes of a retrospective*. In our terminology, a *team-*

level retrospective refers to a retrospective conducted by the software development team members, such as sprint retrospectives or team-level post-mortem analysis (Schwaber and Sutherland 2011; Bjørnson et al. 2009).

We formulated the following research questions with respect to the case organisation:

RQ 1 What discussion topics are covered in the team-level retrospectives?

- RQ 1.1 *What development process areas do the discussions concern*?
- RQ 1.2 *What topic types do the discussions concern*?
- RQ 1.3 *To what degree are the discussion topics related to negative and positive experiences*?

With the first research question, we aim at descriptive results concerning the topics mentioned in the retrospective discussions in the case organisation. The main analytical viewpoints are the process areas and the types of discussion topics. We used and extended a previously developed taxonomy for discussion topics, described in the section 3.5: Data analysis. In addition, we study the amount and types of negative and positive experiences, since retrospectives typically cover both.

RQ 2 For which discussion topics are corrective actions developed?

- RQ 2.1 *What discussion topics most often result in the development of corrective actions*?
- RQ 2.2 *What types of corrective actions are developed*?

With the second research question, we aim to understand what discussion topics attract the most ideas for corrective actions and describe the developed corrective actions using the same analytical framework as for the discussion topics in general in RQ 1.

RQ 3 How do the discussion topics evolve over time?

RQ 3.1 *How do the discussion topics change over time*?
RQ 3.2 *What discussions keep recurring over time*?
RQ 3.3 *What corrective actions are developed for the recurring discussions*?

With the third research question, we aim to understand the longitudinal evolution of the retrospective discussions when the retrospectives are applied as continuous practice over a period of several years. Retrospective meetings should be applied as a continuous and frequent practice by development teams. We investigate how the discussion topics evolve over time and identify the changes in the topic types and the process areas. To obtain more insight into how often the retrospective discussions recur, we analyse to what extent the retrospective meetings repeat the same discussions and analyse in detail the types of discussions that keep recurring and the related corrective actions.

RQ 4 How well do the retrospective discussions correspond to the development repository data?

The fourth research question focuses on how well the retrospectives reflect the actual development status in the organisation. We investigate the available software development repository data to obtain insight into the development status. We focus in our analysis on the recurring retrospective discussions because we were able to analyse longitudinally those particular discussions for the duration of the study timeline and compare the changes in the discussions to the changes in the repository data. In this way, we were able to conduct a more reliable analysis than if we had focused on snapshots of a single or a few retrospectives.

Based on this empirical analysis we aim to state implications and research hypotheses regarding the team-level retrospective practices and the contribution of the retrospective practice to organisational knowledge creation and process improvement (see section 5).

3.2 The Case Organisation

Our research focuses on the team-level sprint retrospective practices in a distributed development organisation consisting of approximately 30 employees. The total size of the case company is 800 employees. The organisation develops complex software systems integrated into customer-specific software with varying business logic and into customised hardware provided by the company's partners. The study can be classified as a holistic single case study because the case and the unit of analysis are the same (Yin 1994).

The case was selected based on our stated research goals. The main goals guiding case selection were, first, to gain access to the continuous team-level retrospective practices and their longitudinal outcomes and, second, to study the large-scale context in contrast to a single team setting. We selected a revelatory case (Yin 1994), which enabled us to study a yet unstudied phenomena. This case enabled us to study, over a long period of time, the outcomes of continuous team-level retrospectives conducted by distributed Scrum teams in a complex software engineering context. The case setting provided us with access to an industrial real case setting with a relatively long history of recorded data on continuous team-level retrospectives that were carried out using similar methods as those applied in our earlier research. This is a rarely studied empirical context for retrospective studies and, as such, the research dataset is unique.

The organisation's representatives informed us that they were using the Scrum method (Schwaber and Sutherland 2011), which they had introduced about a year prior to the start of our study. At the same time they started to use the Scrum method, they also started to collect the outcomes from the retrospectives, data that was later provided to the researchers.

The development organisation includes software developers, lead developers, Scrum masters and product owners. Though the skillsets of the developers varied slightly, they did not have any specific developer roles; instead, the UI design and architecture were handled, for example, by the cross-functional development teams. The organisation also included testers who were not part of the development teams. From its very start, the organisation has distributed software systems to three European countries, each having one local product owner and several software developers. The product owners convey the customers' needs to the developers, which enables quick face-to-face collaboration for those working in the different development roles, if needed.

The organisation grew during the 3 years of using Scrum, a major challenge for software development work. During the first year, the developers worked in one distributed software development team (referred to as *stage 1* in the results and analysis section of this paper). Later, in the beginning of the second year, the developers were divided into two distributed

teams (*stage 2*). Finally, during the third year the developers were divided into six country-specific, distributed teams (*stage 3*). In this study, we collected retrospective outcomes from seven individual teams, including all teams in stage 1 and stage 2 and four teams in stage 3. The stages of organisational growth and the retrospectives are illustrated in Table 1.

The release cycle of the organisation is 1 month and the teams use two-to-four-week development sprints. They also conduct daily stand-ups, sprint demonstrations and retrospectives. In stage 1, the length of the sprints varied between 2 and 3 weeks (development and testing work were integrated). In stage 2, the sprints were lengthened into 3 weeks (development and testing work were integrated). In stage 3, the sprints started with 3 weeks of development work and continued with 1 week of testing work (see the timeline of the retrospectives, sprints and stages of the team structure in Table 1).

3.3 The Retrospective Methods Used in the Case Organisation

The case organisation used a continuous retrospective approach for making improvements in its software development work at the team level.

The team-level retrospectives were conducted at the end of the software development sprints (Schwaber and Sutherland 2011) along with sprint demonstrations and planning events. The retrospectives were one-hour-long, face-to-face meetings conducted separately in each development team. The results of the retrospectives were used for engaging in continuous reflection and making process improvements in the software development practices. The retrospective meetings were attended by all available team members, which meant approximately 3–5 people in each meeting, and they were facilitated by the Scrum masters. However, the product owners did not regularly participate in the retrospective meetings.

The retrospective practices followed the principles of post-mortem analysis and consisted of three main themes: positive experiences, negative experiences and improvement ideas. The focus was on the experiences from the latest development sprint. First, the positive experiences, successes and achievements were identified and recorded onto a diagram. These findings and their relationships were discussed. Second, the negative experiences and perceived problems were identified and recorded onto the diagram. The participants were additionally asked to express the causes for their findings by using root cause analysis (RCA), as suggested by (Bjørnson et al. 2009). During the RCA phase, the team members first listed underlying causes onto the diagram. Then, they discussed the findings and tried to detect deeper level causes. Third, each participant voted on the most important improvement targets. Fourth, corrective actions were developed for the target problems that received the most votes. Finally, the improvement ideas were discussed.

In all of the retrospectives, a retrospective support tool was used for collecting and recording the findings onto the RCA diagram in collaborative manner. The outcomes of each

Table 1 Research timeline showing the stages of the organisation's team structure, the sprint lengths and the retrospective dates

Sprint length	~2-3 weeks														~3 weeks															~3 weeks + 1 week integration test							
Teams	Stage 1: 1 team														Stage 2: 2 teams															Stage 3: 6 teams							
Sprint number	0	2	3	4	5	6	7	8	9	10	12	13	14	15	16	17	18	19	20	20	21	21	22	22	23	23	24	26	26	27	28	28	28	28	30	31	37
Retrosp. number	0	2	3	4	5	6	7	8	9	10	12	13	14	15	16	17	18	19	20a	20b	21a	21b	22a	22b	23a	23b	24	26a	26b	27	28a	28b	28c	28d	30	31	37
Retrosp. date	23-Dec-11	28-Feb-12	20-Mar-12	11-Apr-12	08-May-12	22-May-12	05-Jun-12	19-Jun-12	07-Aug-12	21-Aug-12	03-Sep-12	02-Oct-12	23-Oct-12	13-Nov-12	04-Dec-12	08-Jan-13	25-Jan-13	15-Feb-13	08-Mar-13	08-Mar-13	28-Mar-13	28-Mar-13	22-Apr-13	22-Apr-13	17-May-13	17-May-13	10-Jun-13	30-Aug-13	30-Aug-13	27-Sep-13	24-Oct-13	25-Oct-13	25-Oct-13	25-Oct-13	31-Jan-14	28-Feb-14	24-Sep-14

retrospective were used for making improvements in the development organisation, but the recorded outcomes of the previous retrospectives were not used as input for the forthcoming retrospective meetings.

3.4 Case Study Methods and Data Collection

Data collection and analysis were carried out in multiple steps. The data consist of two datasets collected by the team members as part of their normal team retrospective and development work over the course of 3 years, 2012–2014. The datasets were retrieved at the end of 2014 and analysed in 2015.

The first dataset consists of documented outcomes from 37 team-level sprint retrospectives. This dataset was collected by the company personnel and the data collection process was carried out by using a retrospective support tool, the ARCA-tool,[1] during the retrospective meetings. Figure 2 provides an example of the collected data. The retrospectives were facilitated by the Scrum masters, and the authors were not present during these meetings. All of the retrospective meetings from which data are available are listed in Table 1.

The team members used the software tool to register their findings and collaborate with other team members during the meeting. The tool provided the team members with features to record the retrospective statements onto a diagram, form relationships between the statements, vote for particular statements and record corrective actions. In addition, the tool made it possible for them to organise and classify the statements directly onto the created diagram. All participants can access the same diagram simultaneously and collaboratively. The tool does not affect the retrospective practices to any significant degree. It does not force users to apply any of the features, and in the case organisation each retrospective was started with an empty diagram. This tool was developed as part of our earlier research project (Lehtinen et al. 2014b), and the tool is in constant use, even though it is still not widely used in the industry.

The second dataset includes the task repository of the case organisation. The data were collected by the authors to evaluate whether the retrospective outcomes, especially the recurring discussion topics, were reflected in the company's task repository. The company was using Jira[2] to manage its development tasks. We collected information from the repository on the development tasks implemented over the three-year study period. We studied the organisation's defects and task backlogs in order to analyse the number of open defects and the estimation accuracy.

In addition to the primary datasets, we collaborated with the company's representatives during the study. We interviewed one of the Scrum masters in order to better understand the software development activities. This interview was not used directly as data to answer the research questions. It confirmed and detailed the researchers' understanding of the case organisation, its work practices and the evolutionary stages and major milestones in the organisation over the three-year timeline of the study. We also introduced some of the preliminary findings of the analysis to the company's representatives in order to validate the results.

The data collection methods were selected to take advantage of the best possible evidence from the retrospective meetings over a long period of time. We could not rely on the participants' recollections regarding activities that had taken place a few years back. The

[1] https://github.com/WiRCA/ARCA-tool
[2] Issue and project tracking tool (https://www.atlassian.com/software/jira)

researchers were not present in the retrospective meetings and the meetings were not audio or video recorded. This study and its results rely purely on the documented outcomes that the team members recorded during the meetings.

The data analysis followed the steps outlined in Fig. 1. First, the raw retrospective data were collected and saved into a custom database that was created for the purposes of analysis. The raw data included the contents of the retrospective diagrams from the company sprint retrospective meetings. All individual retrospective statements were imported into the database, with reference to the particular retrospective meeting that the statements originated from. All data that were directly available in the raw diagrams was saved. This included the number of votes given and the classification of negative statements, positive statements and corrective actions as well as the relationships between the retrospective statements (see section 3.5.1).

Second, the recorded retrospective statements were coded by using and extending an existing coding scheme. Based on the coding categories, a quantitative analysis was performed with respect to the topic types and process areas that were discussed in the retrospectives and the types of topics and areas for which corrective actions were developed.

Third, the coding and quantitative analysis was used to create a synthesis of the all the discussion topics and their interrelationships. The statement coding and quantitative analysis, together with the synthesis of the relationships, were used to answer research questions RQ 1 and RQ 2.

Fourth, all retrospective statements were connected to a timeline based on the retrospective meeting dates. The timeline was used to analyse the evolution of the retrospective discussions during the three stages of the case organisation's history. This analysis was used to answer RQ 3.

Fifth, after connecting the retrospective data to the timeline, the recurring discussions were identified. This was done by qualitatively identifying similar statements in each discussion

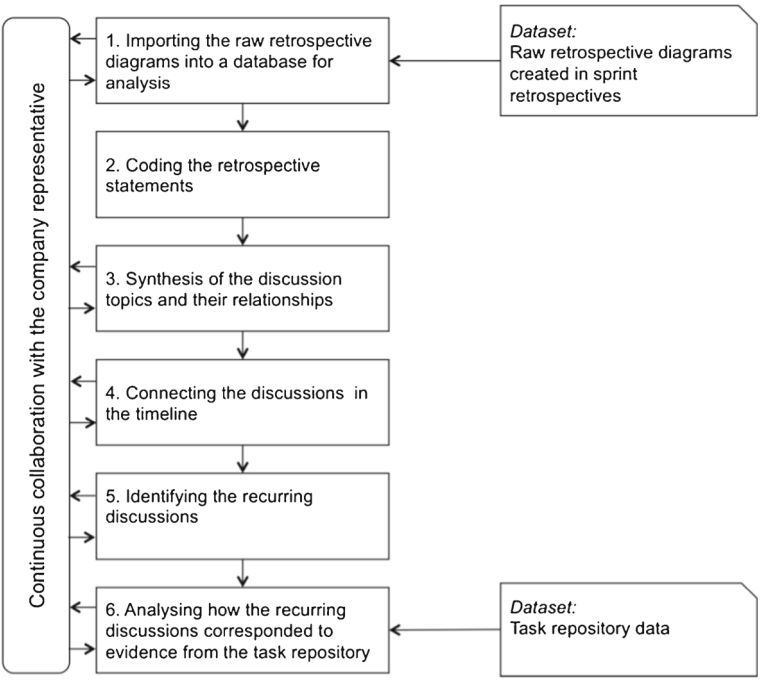

Fig. 1 Research methodology overview illustrating the research steps, methods and datasets

topic category. Discussions occurring in more than 10 % of the retrospective meetings were selected for further analysis. The share of recurring discussion topics and the developed corrective actions were analysed quantitatively. In addition, the discussions were qualitatively analysed based on the statement texts. How often the discussions recurred was illustrated by connecting the number of times a statement was repeated during discussions to the timeline. Our analysis of recurring discussions was used to answer RQ 3.

Sixth, after identifying the recurring discussions, these particular discussions were validated by comparing the contents of the recurring discussions to the task repository data based on the number of times the individual statements occurred on the timeline. This analysis was used to answer RQ 4.

The first two authors were involved in all research design and data analysis activities throughout the study, working as a pair. The first author coded the raw data, step 2 in Fig. 1. Both the first and second author were involved in collaborating with the case organisation. The third author had more of a supporting and commenting role in the data analysis process and contributed more to commenting on and writing the manuscript.

3.5 Data Analysis

We analysed the data using both quantitative and qualitative methods. Section 3.5.1 describes the research data and presents the concepts and categorization system that we used to characterise the raw data and compare the various retrospectives. Section 3.5.2 discusses how the quantitative analyses made use of the categorisation system to summarise and compare the number of discussion topics raised during the retrospectives. Finally, section 3.5.3 describes the methods applied in the qualitative analysis.

3.5.1 Retrospective Data and the Categorisation System

The raw data from the retrospective meetings included *retrospective statements*, which were arranged into a cause-effect diagram (see Fig. 2 for an example of such a diagram) in the retrospective meetings. A retrospective statement refers to the raw data before coding or

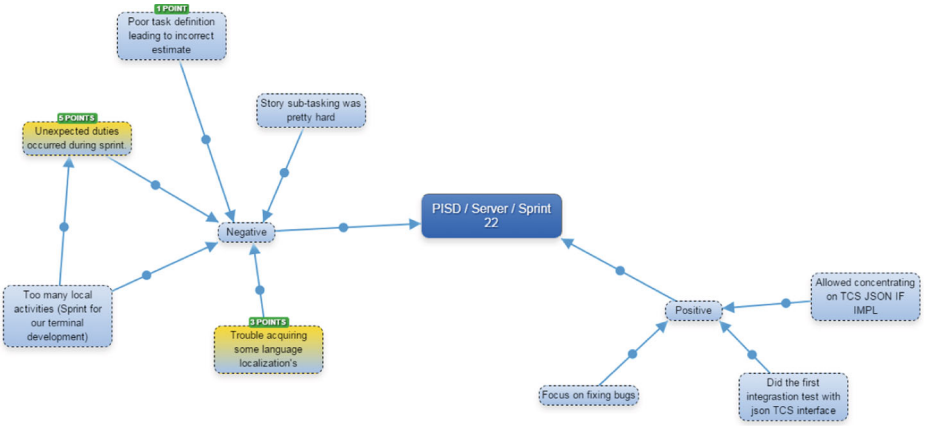

Fig. 2 An example of an output diagram from a team-level retrospective, created using the retrospective tool in the case organisation. The diagram includes a cause-effect diagram, voting results ('points') and corrective actions (registered into the yellow-coloured nodes)

interpretation by the researchers. Figure 3 presents the main concepts related to the research data and the categorization system. The main concepts are as follows:

- *Retrospective statement* – refers to the exact verbatim statements that the retrospective participants recorded into the diagrams. There are four data points for each statement, all of which are directly available from the raw diagrams: 1) the statement text, 2) the number of votes the statement received in the meeting, 3) the outcome class; negative statements, positive statements or corrective actions, and 4) the relationships (links) between the statements. The relationships include statement-to-sub-statement relationships and statement-to-corrective action relationships.
- *Process area* – each retrospective statement belongs to one process area (see Table 2).
- *Topic type* – each retrospective statement belongs to one type of topic (see Table 3).
- *Discussion topic* – each retrospective statement belongs to one discussion topic, which is a combination of the *process area* and *topic type* of that statement.

A categorisation system was used to analyse the topics discussed in the retrospective meetings. The categorisation system consists of two dimensions, *process area* and *topic type*. Together, these two categories characterise the *discussion topic* of each retrospective statement. The *process area* expresses where in the software development process that the retrospective statement is referring to, eg 'sprint planning' or 'implementation work' (see Table 2). The *topic type* characterises the object of the statements, eg 'learning' or 'task estimations' (see Table 3). Thus, the combination of the process area and the type categories represents the discussion topic, eg 'task estimations in the implementation work'. The discussion topic is the concept that we study in this study and the process-area and topic-type categorisations are used to operationalise the concept and make it possible to analyse a large volume of natural language data.

We used the categorisation system introduced in our prior work (Lehtinen et al. 2014a, 2015b) as a starting point for developing the categorisation system used in this study. The categorisation system has been used to state hypotheses regarding the factors affecting the outcome of software projects (Lehtinen et al. 2014a). The categories reflect the general factors

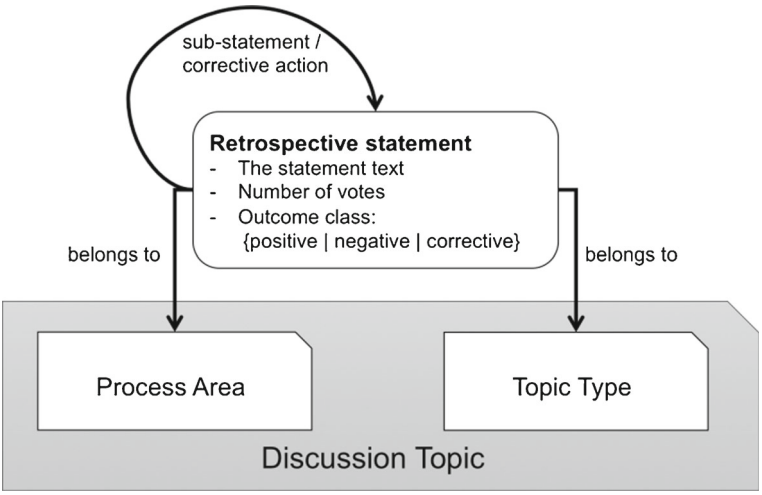

Fig. 3 The concepts used to categorise and analyse the retrospective data

Table 2 Process-area categorisation

Process Area	Characterisation	Sample statements
General Management	Company support and the way the project stakeholders are managed and allocated to tasks.	*Still a lot of ad-hoc tasks* *Unexpected duties occurred during sprint*
Sales & Requirements	Requirements and input from customers.	*Lack of accurate specifications* *Specifications obtained too late*
Product Owner	The product owner's actions.	*PO did not have enough time* *PO answer's not available*
Scrum Master	The Scrum master's actions.	*SM is busy* *SM has difficult role combination*
Sprint Planning	Planning work, including estimations, prioritisation, task descriptions and scoping	*All developers did not participate to the planning* *Bad planning during sprint planning*
Implementation	The design and implementation of features, including the fixing of defects.	*Daily Scrums are not started on time* *Install scripts were broken*
Software Testing	Test design, execution and reporting.	*Problems with dev-test held us back for 2 days* *Bugs often lack information that is necessary*
Deployment	Releasing and deploying the product.	*Old code in production* *Issues in CI (continuous integration server)*
Personal Life	Everything that is related to personal life outside the company.	*Holiday season before sprint* *Lot of contacts outside the company while part time.*
Unknown	Topics that cannot be focused on in any specific process area.	*People too busy to help* *Too many meetings*

affecting the outcome of software system development (McLeod and MacDonell 2011), and the system has been presented as feasible for describing how the factors are related to one another (Lehtinen et al. 2014a). For this study, we extended the existing categorisation system by adding new process area categories to it. This was done iteratively by categorising the outcomes of the retrospectives into the existing system and modifying the system if needed.

3.5.2 Quantitative Analysis

For the quantitative analysis, we categorised the retrospective statements according to the categorisation system. This categorisation was then used to analyse and compare the topics raised during the retrospective discussions. During the retrospective meetings, the team members divided the retrospective statements into three *outcome classes*: 1) positive software development experiences, 2) negative software development experiences and 3) corrective action. Thus, the retrospective statements in the data were divided into these three classes and the number of retrospective statements were analysed as a whole as well as in the three outcome classes.

After recording all of the raw data into a database, the researchers qualitatively coded (the arrows in Fig. 3) each retrospective statement by applying two categorisation dimensions: topic types and process areas. There were occasionally retrospective statements that included two sentences with a different process area or topic type in the same raw statement. In these

Table 3 Topic-type categorisation

Topic type / Sub-type	Characterisation	Sample statements
People (P)	This topic type includes statements regarding human aspects	
P1 Instructions	Documentation and instructions	*Lack of information about stories* *I was able to acquire all necessary info to complete the story*
P2 Experience	Skills and experience	*Still lacking knowledge of the entire system* *We managed without X (name of the key person removed)*
P3 Learning	Learning new techniques, etc.	*Lack of training on how to use them* *Learned a lot of new stuff*
P4 Values & Responsibilities	People's attitude and taking responsibility	*Sprint work is slow to start* *People do not focus on daily* [*Scrum*]
P5 Cooperation	Collaboration and communication	*Not enough help available* *Got enough help*
P6 Motivation	Motivation to act	*Frustration over inexperience and too vague story guidelines* *Enjoyed doing story*
P7 Policies	Following the rules of the organisation	*Meeting room was not available all time despite the reservation* *Daily scrums are not started on time*
Tasks (T)	This topic type includes the task-related statements	
T1 Task Monitoring	Monitoring of the task progress	[*Difficulties in*] *monitoring and QA things* *Some monitor scripts to check that X usage is unified*
T2 Task Priority	The priorities of the work	*Priority changes* *Focus on fixing bugs*
T3 Task Allocation	The allocation of the work	*Unclear roles – Who is responsible for what*? [*There were*] *clear roles and own tasks for everyone*
T4 Task Outcome	The quality and concrete outcome of work	*A lot of bugs* *Task definition was clear*
T5 Task Difficulty	The required effort, or time, or task complexity	*Hard to plan for this sprint* *Lots of easy bugs to fix*
T6 Task Risk	The risks related to the work	*A lot of potential bugs can occur, even when implementing simple message transformation*
T7 Task Progress	The progress of the work items	*Had to remove one story from sprint* *We managed to get bug count low*
T8 Task Missing	Tasks that are missing from the work	*Translation of the new localisation keys is not a task in a sprint* *No planning was done for this*
T9 Task Estimations	The effort and schedule estimations for the work	*Efforts were not accurate* *Completed tasks had accurate estimates*
Methods (M)	This topic type includes the methodological statements	
M1 Process	Process of the related work	*We do not have any localisation process* *Process works pretty smoothly*

Table 3 (continued)

Topic type / Sub-type	Characterisation	Sample statements
M2 Work Practices	Practices used to conduct the work	*Developer should test that the implementation fulfils the requirement* *Code reviews went well*
M3 Tools	Software tools used in the work	*X does not have any good plugins for Y* *X helped a lot*
Environment (E)	This topic type includes statements about the environment	
E1 Existing Product	The existing product that was implemented	*Very large system that takes time to learn* *Quality debt in the software*
E2 Resources & Schedules	The available resources and given schedules	*Sick leaves* *Finished sprint in time*
E3 Customers & Users	Customers' and users' expectations and needs.	[*Customers*] *do not follow standards* *Lacking support & info from* [*customers*]

cases the statement was split into two statements during the analysis in order to categorise it correctly. The first author coded the raw data using the categorisations.

We compared the shares of the statements across the topic types and process areas. This enabled us to characterise the common discussion topics for the retrospective outcome classes and recognise changes in the discussion topics over time. We identified the discussion topics that were commonly presented as a target for corrective actions and compared the number of corrective action targets and actual corrective actions for the topic-type and process-area categories.

We analysed how the discussion topics evolved over the retrospective timeline. We grouped the retrospectives into three stages based on the team structures at the time that the retrospectives were conducted. The stages are stage 1 (first year, one team), stage 2 (second year, two teams) and stage 3 (third year, many teams). We compared the distributions of discussion topics over the stages in order to recognise any possible change in the discussion topics as the organisation grew.

We studied how often the discussions recurred in the retrospectives. After qualitatively recognising the recurring discussions (see section 3.5.3), we analysed the number and frequency of the recurring discussions.

In addition to the retrospective statements, we used the task repository data to analyse measurable changes in the bug counts and estimation accuracy. We compared the task repository data with the recurring retrospective statements regarding high or low bug counts and poor or successful task estimates. This comparison was done during the specific time periods when the related retrospective discussions occurred. Regarding the bug counts, we used the number of open bugs as the measurement. The changes in the bug count were analysed using a line chart over time. Regarding estimation accuracy, we measured the share of the reported effort per the estimated effort for each task in a development iteration. The changes in the estimation accuracy were analysed using a box plot.

3.5.3 Qualitative Analyses

We used qualitative techniques to synthesise how the retrospective statements were related throughout the process areas. Due to the use of root cause analysis in the retrospectives, the raw data included perceived cause-effect relationships. We explored the cause-effect

relationships in the raw data and drew conclusions regarding how they were interconnected in the process areas.

We also used qualitative techniques to recognize whether a retrospective statement occurred continuously, which enabled us to develop our hypotheses regarding the recurring discussions. We also analysed the corrective actions in order to recognise whether a corrective action was developed multiple times. Furthermore, we analysed the corrective actions developed for the recurring discussions in order to characterise how the team members tried to resolve them.

We began the analysis by focusing on the discussion topics that were continuously covered in the retrospectives. The categorisations were used as a technique for selecting the retrospective statements (ie the raw data) for further investigation. We considered all statements, which were categorised into same discussion topic (process area and topic type). We analysed each discussion topic category in order to recognize similar, recurring statements by exploring every statement in the discussion topic category. The recurring statements were combined to create a comprehensive description of the recurring discussions. Finally, the recurring statements were mapped onto the retrospective timeline in order to visualise how often the recurring discussions were occurring during the retrospectives.

4 Results and Discussion

This section presents our results together with the discussion. The section is structured according to the four main research questions (see section 3.1). The results are presented for each research question, followed by a discussion of the results for each main research question under the reflective subsections.

The analysis is based on the raw statements that were directly recorded in the retrospective meetings by the participants using a software tool. The statements, in the raw data, were classified into negative experiences, positive experiences and corrective actions. In addition, the data includes interrelationships between statements and the number of votes each statement received in the retrospective meeting. The quantitative analysis is based on our categorisation of the retrospective statements into process areas and topic types. The combination of process area and topic type is called the discussion topic in this article. See section 3.5.1 for a detailed description of the concepts and classifications used in the analysis.

4.1 What Discussion Topics are Covered in the Team-Level Retrospectives? (RQ 1)

This section presents our analysis of the topics that the team-level retrospectives in the case organisation discussed during the study period. We divided the analysis into three sub-questions regarding the process areas that the retrospective statements concern, the topic types and the degree of positive and negative discussions. Table 4 presents a summary of the most common positive discussion topics, negative discussion topics, discussion topics that the participants voted for and corrective actions. In the table, common discussion topics (the combination of process area and topic type, see Fig. 3) are described and illustrated together with concrete examples from the retrospective statements. The *progress of implementation work* was one of the most common discussion topics. The *tools* and *resources & schedules for the implementation work* were also commonly discussed both positively and negatively, but they did not result in many related corrective actions. *Learning about the implementation work* was mostly discussed based on positive experiences and rarely mentioned in the negative, and

Table 4 The most common discussion topics in each outcome class (columns). Each table cell includes, first, the *topic type* and *process area*. After that, the numbers in parentheses indicate the total number of times that each discussion topic occurred, or received votes, in all team-level retrospectives. The quotes are examples of specific statements

#	Positive discussion topics (n of statements)	Negative discussion topics (n of statements)	Voted discussion topics (sum of votes)	Corrective actions (n of statements)
1	Progress of implementation work (47), eg '*we managed to fix lots of bugs*'.	Tools for implementation work (29), eg '*problem to connect tools what we are using*'.	Outcome of planning work (26), eg '*too big stories*, *or even epics* (*as story*), *taken into sprint*'.	Improving the planning work practices (20), eg '*estimate bigger numbers and/or accept less*'.
2	Resources & scheduling of implementation work (19), eg '*sufficient workload*'.	Resources & scheduling of implementation work (21), eg '*too much to do*'.	Resources & scheduling of implementation work (20), eg '*too short sprint*'.	Improving the implementation work practices (14), eg '*resolve problems after daily* [*stand up meeting*] *not in it*'.
3	Learning about implementation work (19), eg '*good knowledge transfer*'.	Task estimations for implementation work (21), eg '*bad estimations*'.	Tools of implementation work (20) eg "*Crucible license is going to expire*"	Improving the implementation work instructions (12), eg'*some basic documentation as Javadoc*'.
4	Collaboration in implementation work (19), eg '*I got help when I needed*'.	Outcome of planning work (19), eg '*no use case in story*'.	Instructions for implementation work (19), eg '*information and documentation on classes/tasks could be improved*'.	Improving the software testing work practices (10), eg '*test more on dev-test*, *if updating preprod more often not feasible*'.
5	Outcomes of planning work (17), eg '*good split up of sub-tasks*'.	Work practices of implementation work (18), eg '*not everyone is using IntelliJ Idea*[2] *yet*'.	Task estimations for implementation work (14), eg '*estimating larger stories effort still quite inaccurate*'.	Improving the planning work outcomes (6), eg '*unexpected duties should be taken in account when timeboxing and estimating sprints*'.
6	Task estimations for implementation work (13), eg 'good estimates'.	Progress of implementation work (18), eg '*could not complete our tasks*'.	Experience with implementation work (13), eg '*still lacking knowledge/overview of the entire system*'.	Improving general management's work practices (5), eg '*no sit-down meeting should be held without prewritten agenda*'.
7	Work practices of implementation work (11), eg '*code reviews went well*'.	Instructions for implementation work (15), eg '*no documentation for new developers*'.	Outcomes of software testing (13), eg "'*contact < name >*[1] *before fixing this*" *in bugs description*'.	Improving product-owner collaboration (5), eg '*improve communication with product owners about priorities*'.
8	Tools for implementation work (10), eg '*IntelliJ Idea*[2] *works*'.	Tools for software testing (13), eg '*no SVN checks still for code comments and check style errors*'.	Progress of implementation work (11), eg '*new feature implementation could not be finished*'.	Improving the work planning process (5), eg '*go through session before sprint planning*'.
9	Outcomes of implementation work (9), eg '*low bug count*'.	Outcomes of sales & requirements (11), eg '*requirements are not specific enough*'.	Experience with planning work (10), eg '*still not enough experience with planning tasks*'.	Improving the implementation work resources & scheduling (5), eg '*reserve time in sprint to pay for the quality debt*'.
10	Progress of product deployment (8), eg '*1.21 in production*'.		Outcomes of sales & requirements (9), eg '[*we need*] *better specifications*'.	

Table 4 (continued)

#	Positive discussion topics (n of statements)	Negative discussion topics (n of statements)	Voted discussion topics (sum of votes)	Corrective actions (n of statements)
		Task difficulty of implementation work (11), eg '[*the client system*] *is annoying and takes time*'.		Improving the implementation work process (5), eg '*have a process that you update documentation when you modify code*'.
11	Task priorities for implementation work (7), eg '*focus on fixing bugs*'.	Outcomes of software testing (11), eg '*bugs often lack information that is necessary*'.	Outcomes of general management (9), eg '*large amount of urgent, unplanned work*'.	Improving the implementation work tools (5), eg '*add the check-style plugin to eclipse*'.
12	Tools for software testing (5), eg '*testing connection is now set up, which allows transaction testing*'.	Task priorities for implementation work (10), eg '*priorities are not clear*, ie *several blockers*'	Task allocation for implementation work (9), eg '*personal allocations do not hold*'.	Improving the software testing outcomes (5), eg '*describe more detailed steps in how to reproduce*'.
13		Outcome of implementation work (10), eg '*almost every issue had several bugs*'.	Outcome of implementation work (9), eg '*a lot of bugs*'.	

[1] Person's name not disclosed

[2] Refers to a development tool (https://www.jetbrains.com/idea/)

it resulted in few corrective actions. *Software testing tools* was commonly discussed based on negative experiences and rarely mentioned in the positive, and it did not result in many related corrective actions. Furthermore, the (lack of) *experience with implementation work* was a common discussion topic, one that the participants voted as important, but still it was rarely discussed and rarely formed the basis for the presented corrective actions. Most corrective actions were related to *work practices*.

4.1.1 What Development Process Areas do the Discussions Concern? (RQ 1.1)

The total numbers of retrospective statements recorded during the whole research period are summarised in Table 5. The table also shows the distribution of the statements across process areas and with respect to the positive, negative and corrective action classes. The data reveal that the discussions included statements regarding almost all process areas. The bulk of the statements concern the areas of sprint planning, implementation work and software testing, together comprising 75 % of all statements.

Figure 4 shows *how often* statements regarding each process area were mentioned in the retrospective meetings. The diagram indicates the percentage of all retrospectives where at least one statement was made regarding a certain process area. All retrospectives included statements related to implementation work. Statements concerning sprint planning and software testing were detected in more than 60 % of the retrospective meetings, meaning that the frequency distribution follows a similar profile over the process areas as the total numbers of statements.

Figure 5 shows *how much* each process area was discussed in the retrospective meetings. The diagram indicates the average number of statements in the retrospective meetings in which at least one statement was made regarding a specific process area. Participants allotted a similar amount of attention to all process areas. Implementation work stands out also in this data as a topic that received more emphasis. In addition to implementation work, participants discussed and voiced more positive experiences with the product owner and the sales and requirements areas than they did with other areas.

Table 5 Number of retrospective statements in all retrospectives distributed across each of the process areas

Process area	Positive	Negative	Corrective actions	Total	%
Sales and Requirements	3	28	11	42	5 %
General Management	3	24	10	37	4 %
Product Owner	4	17	18	39	4 %
Scrum Master	0	4	1	5	1 %
Sprint Planning	25	51	40	116	13 %
Implementation work	170	198	60	428	49 %
Software testing	24	59	28	111	13 %
Deployment	16	32	8	56	6 %
Unknown	9	32	4	45	5 %
Total	254	445	180	879	
%	29 %	51 %	20 %		

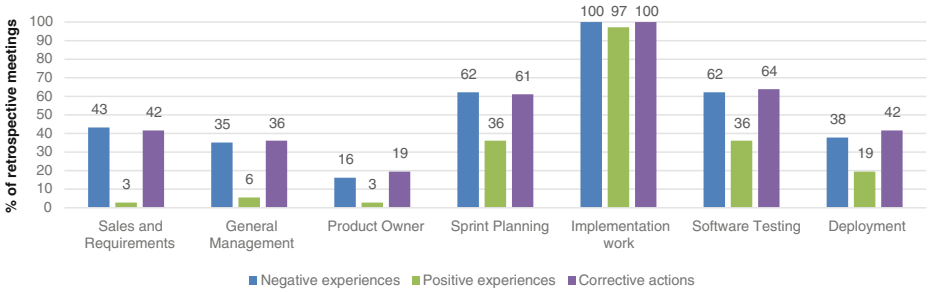

Fig. 4 The occurrence of retrospective statements in meetings for each of the process areas (% of retrospective meetings)

4.1.2 What Topic Types do the Discussions Concern? (RQ 1.2)

The distribution of the retrospective statements across each of the topic types is presented in Table 6. The data show that the topic types most often discussed were work practices, task outcome and progress, and resources & scheduling, as well as cooperation, instructions and learning. The least discussed topics were task risks, task monitoring, policies and customers & users. The task types (denoted with the letter T in Table 6) were most common in both positive (45 %) and negative (39 %) statements, whereas the types of methods (denoted with the letter M) represented almost half (49 %) of the corrective actions.

Figure 6 presents a synthesis of the retrospective outcomes. The three most common process areas in the discussions were closely interconnected to one another with respect to perceived cause-effect relationships, which indicates that the retrospectives also included discussions on the workflow related to planning, implementation and testing. Interestingly, discussions about the general management, the product owner and deployment were disconnected from the other areas.

4.1.3 To What Degree are the Discussion Topics Related to Negative and Positive Experiences? (RQ 1.3)

Overall, negative statements were more common than positive statements, as the total number of positive statements in retrospective discussions comprised only 57 % the total number of negative statements (see Table 5). All retrospectives included negative experiences related to the implementation work. The detection of negative experiences was frequent in other process areas, too. In contrast, the detection of positive experiences was more frequent for

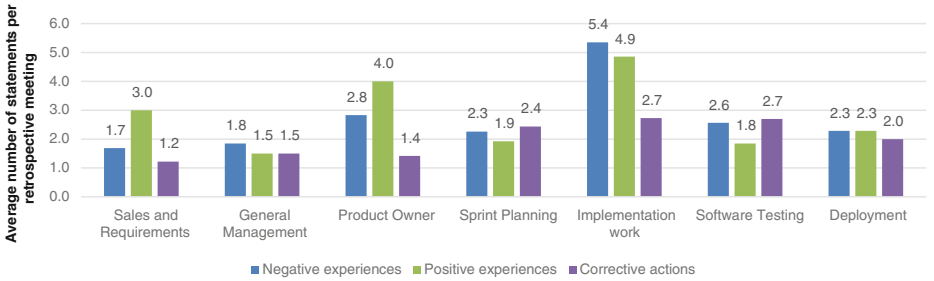

Fig. 5 The average number of statements per retrospective for each of the process areas

Table 6 Number of retrospective statements in all retrospectives distributed across each of the topic types (see Table 3)

Topic type	Positive	Negative	Corrective actions	Total	%
M2 Work Practices	13	57	57	127	14 %
T4 Task Outcome	32	72	18	122	14 %
E2 Resources & Scheduling	28	54	16	98	11 %
M3 Tools	18	57	14	89	10 %
T7 Task Progress	56	27	0	83	9 %
P5 Cooperation	28	13	18	59	7 %
P1 Instructions	3	23	16	42	5 %
P3 Learning	32	5	3	40	5 %
T9 Task Estimations	13	21	2	36	4 %
T5 Task Difficulty	4	29	0	33	4 %
P2 Experience	2	22	2	26	3 %
M1 Process	1	7	17	25	3 %
T2 Task Priority	7	13	3	23	3 %
E1 Existing Product	4	14	0	18	2 %
P4 Values & Responsibility	3	7	6	16	2 %
P6 Motivation	7	5	0	12	1 %
T3 Task Allocation	2	4	3	9	1 %
T8 Task Missing	0	4	3	7	1 %
E3 Customers & Users	1	4	0	5	1 %
P7 Policies	0	3	1	4	0 %
T1 Task Monitoring	0	3	1	4	0 %
T6 Task Risk	0	1	0	1	0 %
Total	254	445	180	879	
%	29 %	51 %	20 %		

implementation work than for the other process areas (see Fig. 4). The distribution of the positive and negative statements is quite similar throughout the process areas, with the exception of the sales & requirements, general management and product owner areas, each of which received only a few positive statements in the retrospective meetings (see Table 5 and Fig. 4). Fig. 5 reveals, however, that those rare positive statements led to similar amounts of discussion as did other areas brought up in the meeting.

Regarding the topic types, the most positive topics were task outcome, task progress, cooperation and resources & scheduling. These topics were also commonly focused upon in the negative statements, possibly reflecting a certain level of satisfaction when a team succeeds in activities that the team members feel they frequently struggle with. The topics of task progress, learning, cooperation and motivation all received more positive than negative statements; learning especially received only a few negative statements, but it received the second highest number of positive statements. Many purely negative topic types received only a few or no positive statements at all. It seems that the positive statements cover topics that are closely related to a team's progress and achievements in its implementation work, such as task progress and outcomes. Topics that participants discussed frequently, but mostly in the negative, were instructions, task difficulty, process and experience.

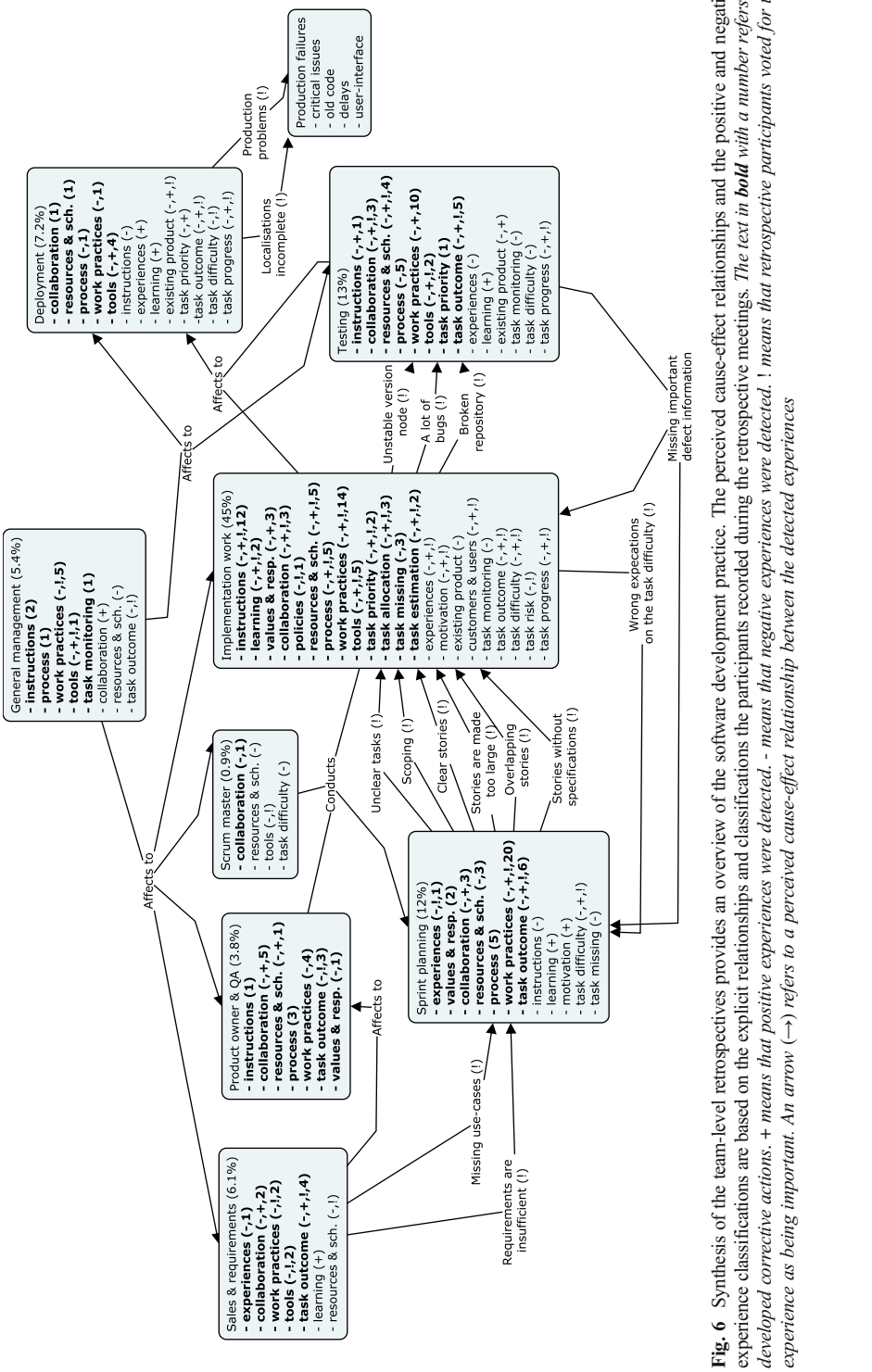

Fig. 6 Synthesis of the team-level retrospectives provides an overview of the software development practice. The perceived cause-effect relationships and the positive and negative experience classifications are based on the explicit relationships and classifications the participants recorded during the retrospective meetings. *The text in **bold** with a number refers to developed corrective actions. + means that positive experiences were detected. - means that negative experiences were detected. ! means that retrospective participants voted for the experience as being important. An arrow (—›) refers to a perceived cause-effect relationship between the detected experiences*

4.1.4 Reflections

Our results show that team-level retrospectives greatly emphasise topics in the process areas close to the interests of the team members. For each outcome class (positive experiences, negative experiences and corrective actions), the discussions were commonly related to sprint planning, implementation work and software testing. This emphasis seems natural for team-level retrospectives since these process areas are closely related to the everyday work of the development teams.

The existing literature has presented a wide array of factors affecting the software development outcomes (McLeod and MacDonell 2011). Prior studies show that the factors are related to the entire software development lifecycle; for example, the factors of better understanding the sales, user and customers (Keil et al. 1998; Moløkken-Østvold and Jørgensen 2005; Drew Procaccino et al. 2002; Cerpa et al. 2010), of determining the requirements (McLeod and MacDonell 2011), and of focusing on quality control and software testing (Jones 2008; Kaur and Sengupta 2011; Egorova et al. 2010). Based on our results, however, it seems that team-level retrospectives are weak at recognising issues and successes related to the process areas that are external to the concerns of the teams (eg sales & requirements, general management and the product owner). In our results, the three process areas most closely related to the interests of the teams accounted for 75 % of the retrospective statements. Many of the earlier works are review studies or surveys covering multiple cases and organisations, with the aim of collecting the full variety of factors, which explains their stronger emphasis on factors beyond the scope of the development teams. Such studies have collected data from different organisational levels, thus providing a wider, more comprehensive picture of all the factors that affect project outcomes or failures. Our results, in contrast, provide a descriptive account of the factors that are emphasised when the analysis is conducted at a team level by the development team.

Prior studies have divided software engineering issues into *internal* and *external* factors based on their controllability for the team members (Xiangnan et al. 2010). We have chosen to use the terms *local causes* and *bridge causes* to express the extent to which the factors are dependent upon organisational issues across process areas (Lehtinen et al. 2014a). The team members recognised internal factors based on the planning work, implementation work and software testing. Their findings represent the local causes and bridge causes affecting across these process areas. Such a rich discussion was not conducted for the process areas external to the interests of the team members. In terms of the external process areas, the team members mainly recognised the existing problems and could not identify any positive experiences. Such problems could be better considered in an organisation-level retrospective because a team-level retrospective might not be an effective forum for discussing problems that are not under the team's control. In our earlier study, we presented an analysis of the organisation-level retrospectives and identified the areas of product owner, sales and requirements as well as a lack of communication and bridging relationships to development practices as being important (Lehtinen et al. 2015b).

In terms of the outcome classes, the number of positive experiences was less than the number of negative experiences, suggesting that most of the discussion was related to the negative experiences. Participants emphasised people factors most often when recounting positive experiences, whereas they most commonly mentioned methodology factors with respect to negative experiences and the need for corrective actions. The results presented in section 4.1.3 highlight the fact that the positive retrospective statements emphasise *learning*, *cooperation*, *resources & schedules*, *task outcome* and *task progress*, ie topic types that characterise people factors and the accomplishments of the team. The negative statements, in contrast, emphasise *work practices*, *resources & schedules*, *task outcome*, *tools* and *task progress*, ie topic types that characterise

methodology factors and task failures. Our prior study on software project failures recognised that software engineering problems are equally related to people and methodology factors (Lehtinen et al. 2014a). It also found that when selecting improvement targets, management prioritises people factors, whereas the employees emphasise the methodology factors. The results of this study corroborate these earlier findings, showing that the development teams tend to find people factors as more positive and avoid stating themselves as a target for improvement.

4.2 For Which Discussion Topics are Corrective Actions Developed? (RQ 2)

In this section, we present the results regarding the developed corrective actions. First, we present our analysis of the discussion topics that were most often the subject of corrective actions in the retrospective meetings. Second, we provide a description of the process areas and topic types with respect to the corrective actions themselves.

4.2.1 What Discussion Topics Most Often Result in the Development of Corrective Actions? (RQ 2.1)

Tables 7 and 8 show the number of corrective actions that were developed based on the negative statements for each topic type and process area, respectively.

The corrective actions were mostly developed for topics that received numerous negative statements. Negative statements regarding such topic types as *task outcome*, *work practices*, *task difficulty* and *instructions* received high number of corrective actions. All topic types that received numerous corrective actions also had a large number of negative statements. However, some topic types received a relatively high amount of negative discussion, but only a few, or no, suggestions for corrective actions. These topic types included *cooperation*, *task progress*, *task priority*, *experience*, *existing product* and *tools*.

Overall the voting practice that was applied in the retrospective meetings guided the development of corrective actions. The majority (66 %) of corrective actions were developed based on the negative statements that participants voted for.

We also analysed the distribution of the topic types with respect to the negative *target statements*, ie the negative statements that the corrective actions were developed for. This analysis is relevant since it shows how the corrective actions often represent other types than the actual negative statement the corrective action is developed for. Table 9 illustrates the differences between the targets of the corrective actions and the actual developed corrective actions. Two groups of topic types are highlighted in the table. First, common types of corrective actions that are not commonly listed as the target statements are shown. Second, common target statements that are not commonly listed as corrective actions are shown.

The developed corrective actions were most commonly in types *work practices*, *cooperation*, *task outcome* and *process*. Of these topic types, *task outcome* and *work practices* were also commonly mentioned in the target statements. The topic types of *cooperation* and *process* were virtually non-existent as target statements, even though both represented approximately 10 % of the corrective actions. Similarly, *tools* and *values & responsibility* were clearly less frequently discussed as targets than their share of the corrective actions would suggest.

On the other hand, *task outcome*, *task estimation*, *policies*, and *task difficulty* appeared more frequently as the target statements for corrective actions than as corrective actions themselves.

Table 7 Number of negative statements and corrective actions developed for each of the topic types

Topic type		Number of negative statements	Number of corrective actions developed for the statements
T4	Task Outcome	72	62
M2	Work Practices	57	36
T5	Task Difficulty	29	19
P1	Instructions	23	18
E2	Resources & Scheduling	54	15
T9	Task Estimations	21	15
M3	Tools	57	6
P7	Policies	3	6
P6	Motivation	5	4
E1	Existing Product	14	3
T7	Task Progress	27	2
P2	Experience	22	2
T2	Task Priority	13	2
P3	Learning	5	2
T8	Task Missing	4	2
M1	Process	7	1
E3	Customers & Users	4	1
T3	Task Allocation	4	1
P5	Cooperation	13	0
P4	Values & Responsibility	7	0
T1	Task Monitoring	3	0
T6	Task Risk	1	0
Total		445	197*

*A total number of 180 corrective actions had 197 relations to negative statements

Table 8 Number of negative statements and corrective actions developed for each of the process areas

Process area	Number of negative statements	Number of corrective actions developed for the statements
Implementation Work	198	79
Sprint Planning	51	32
Software Testing	59	25
Unknown	32	15
Product Owner	17	14
Sales and Requirements	28	13
General Management	24	12
Deployment	32	7
Scrum Master	4	0
Total	445	197

Table 9 Comparison of the topic-type distribution of the negative target statements and corrective actions (the target statements = the negative statements that the corrective actions were developed for). The largest differences are highlighted in bold

Topic type		Target statements (%)	Corrective actions (%)
P2	Work Practices	**18.27**	**31.67**
P5	Cooperation	**0.00**	**10.00**
T4	Task Outcome	**31.47**	**10.00**
M1	Process	**0.51**	**9.44**
P1	Instructions	9.14	8.89
E2	Resources & Scheduling	7.61	8.89
M3	Tools	**3.05**	**7.78**
P4	Values & Responsibility	**0.00**	**3.33**
P3	Learning	1.02	1.67
T3	Task Allocation	0.51	1.67
T8	Task Missing	1.02	1.67
T2	Task Priority	1.02	1.67
P2	Experience	1.02	1.11
T9	Task Estimations	**7.61**	**1.11**
P7	Policies	**3.05**	**0.56**
T1	Task Monitoring	0.00	0.56
E3	Customers & Users	0.51	0.00
E1	Existing Product	1.52	0.00
P6	Motivation	2.03	0.00
T7	Task Progress	1.02	0.00
T5	Task Difficulty	**9.64**	**0.00**
T6	Task Risk	0.00	0.00

4.2.2 What Types of Corrective Actions are Developed? (RQ 2.2)

In terms of the process areas, the number of corrective actions followed the overall profile for the number of negative statements. The *product owner* and *sprint planning* areas represent somewhat higher shares, and *implementation* and *deployment* represent lower shares of corrective actions in comparison to the number of negative statements (see Figs. 4 and 5 and Table 5). Table 6 lists the number of corrective actions in each topic type, while Table 7 lists the number of negative statements and corrective actions developed for them in each topic type. The most common corrective actions were *work practices*. Other common corrective actions were related to *cooperation*, *task outcome*, *process*, *resources & schedules*, *instructions* and *tools*. Some topic types received many negative statements, but did not receive many corrective actions. These topic types include *experience*, *existing product* and the topics related to development tasks, ie *task progress*, *task estimation*, *task difficulty* and *task priority*.

4.2.3 Reflections

Our results show that in terms of the process areas, the distribution of corrective actions follows the distribution of negative experiences (see Fig. 4 and Table 5). The most common process areas for corrective actions were *planning work*, *implementation work* and *software testing*.

The corrective actions were commonly developed for certain topic types (see Table 7). These included *task outcome*, *work practices*, *task difficulty*, *instructions*, *resources & schedules*, and *task estimation*. One explanation for why these topics attracted many corrective actions is that the team members perceived these topic types as being controllable. The team was able to change its work practices, improve its information exchange, change its estimation methods and consider the available resources and schedules. On the other hand, it is remarkable that there were many discussion topics that only received a few, or no, suggestions for corrective actions at all, even though there were many related negative statements. These included *cooperation*, *task priorities*, *task progress*, *existing product*, *experience* and *process*. It seems that solving these problems would have required external support (eg collaboration with external stakeholders) and business critical decision making (eg refactoring the existing product and reprioritising the development of new features). This supports the hypothesis that in team-level retrospectives, the participants tend to focus on solving problems that they feel are under the team's control (see section 4.1.4). Some of these problems were also problems that are inherently difficult to solve, such as estimations, or those related to task progress, difficulty and priority, which are more symptoms of underlying causes rather than controllable root causes.

When we compared the target statements that the corrective actions were developed for and the actual corrective actions (see Table 9), we recognised that the distribution of corrective actions did not follow the distribution of target statements. For example, participants often selected negative *task outcome* statements as a target for corrective actions, but the corrective actions did not focus on the task outcome quite as often. Instead, the related corrective actions suggested the need to make changes in the *work practices*, *cooperation* and *process*, topics that ultimately caused problems with respect to the task outcome, but that are separate from the *task outcome* itself. Another example is *cooperation*, which was never a set target. However, 10 % of the corrective actions were cooperation actions. This indicates that in team-level retrospectives, the root cause analysis does not uncover the actual causes of the negative experiences, but deals more with the visible symptoms. However, as the participants possessed knowledge about the causal mechanisms behind the symptoms, they still felt that they were able to develop the appropriate corrective actions without explicating the exact 'root causes' (Lehtinen et al. 2011). As an example, when they were talking about a negative experience regarding poor task estimates, they directly proposed corrective actions for improving the cooperation between product owners and developers, without considering this lack of cooperation as a cause of the poor estimates. This behaviour might be a problem, since we do not know if the implicit inferences regarding the causes are comprehensive or correct.

4.3 How do the Discussion Topics Evolve Over Time? (RQ 3)

We studied how the retrospective discussions evolved over time using two approaches. First, we compared the discussion topics throughout the three stages covered in the study timeline, each of which represents a distinct phase in the case organisation's history (see section 3.2). Second, we identified the recurring topics of discussion persistent throughout the study timeline. The first approach demonstrated at a higher level how the discussion topics changed over time in terms of the process areas and topic types. The second approach yielded a more detailed analysis of the types of discussions that kept recurring over time, how frequently they occurred and the detailed contents of the discussions. Finally, we also identified the corrective actions developed by the participants with respect to those recurring discussions.

4.3.1 How do the Discussion Topics Change Over Time? (RQ 3.1)

Figure 7 shows the most salient changes in the distribution of the retrospective statements over time (see the description of the stages in section 3.2). The figure includes changes where the percentage of statements in a certain topic type or process area changed nine or more percentage points from one stage to the next during the evolution of the case organisation. Regarding the process areas, see Fig. 7a and b. The figure shows that the share of negative statements and corrective actions in the *software testing* process area increased greatly in stage 2, while the share of corrective actions remained high in stage 3. During stage 3, the share of negative statements and corrective actions in the *implementation* area dropped, while the share of statements in the *product owner* area increased. There were no changes of greater than nine percentage points in the shares of positive statements regarding the process areas.

Regarding the topic types, see Fig. 7c and d. The figure shows that the share of negative statements in the *task progress* and *task outcome* types increased during stage 2 and decreased again during stage 3. The share of positive statements regarding task progress followed a similar pattern, whereas the statements on *resources and scheduling* followed the opposite pattern and were more common as negative topic types in stages 1 and 3 and decreased during stage 2 at the same time that positive discussions on *task priority* and *task progress* increased. Other changes in the positive topic types included a decreasing trend with respect to *learning* and a high share of cooperation discussions in stages 1 and 3, with almost no positive cooperation discussions occurring during stage 2. There were no changes of greater than nine percentage points in the share of corrective actions regarding the topic types.

As a summary, stage 1 and 2 were characterised by a clear focus on the implementation work in retrospective discussions. Stage 2 further emphasised software testing as an

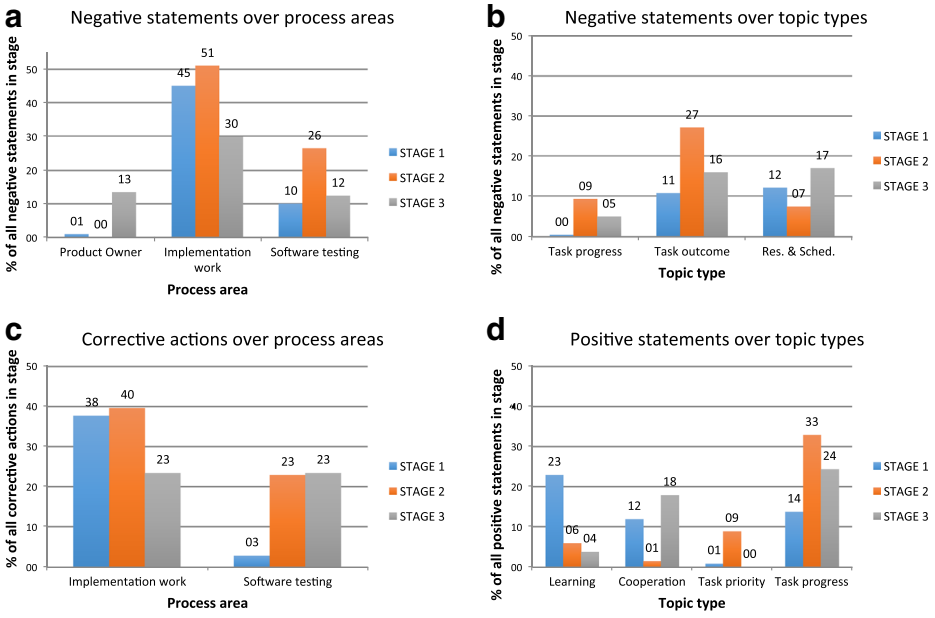

Fig. 7 The most salient changes (9 percentage points or more) in the distribution of statements over the process areas (a and b) and topic types (b and c) in stages 1–3 (no salient changes were observed in terms of the positive statements regarding the process areas or corrective actions regarding the topic types)

improvement area and the discussions focused more on task progress and outcomes and priorities. The task-related topics were common in both negative and positive discussions. Finally, during stage 3 the emphasis on implementation work decreased and testing improvements continued. In addition, task-related discussions decreased, while negative discussions on resources and scheduling and positive discussions on cooperation took place.

4.3.2 What Discussions Keep Recurring Over Time? (RQ 3.2)

We studied how often discussions kept recurring by identifying similar recurring retrospective statements in the team-level retrospectives. A total of 43 individual statements occurred more than once during the study period. The recurring statements were more common when participants discussed positive experiences (23 % were recurring).

The share of recurring statements when participants discussed negative experiences was 9 %. The recurrence of corrective actions was rare — a total of seven actions reoccurred twice, while two actions reoccurred up to three times.

We analysed in detail the ten statements that recurred most often to identify how frequently specific discussions kept recurring in the retrospectives. Each of the ten statements occurred in more than 10 % of the retrospectives. Table 10 summarises the recurring statements that form three distinct recurring discussions. These discussions consisted of the explicit retrospective statements and represent more specific instances than the discussion topics. Figure 8 presents the occurrence distribution for the ten statements that recurred most often throughout the retrospective timeline.

All three discussions included both positive and negative statements. The discussions on the state of bug fixing were related to the number of open defects and how successfully they were fixed. The recurring retrospective statements were as follows: *fixed a lot of bugs* (repeated in 38 % of the retrospectives), *a lot of bugs* (18 %) and *low bug count* (15 %). The discussions on the accuracy of estimations were related to the challenges and achievements in completing the tasks within the sprint schedules. The retrospective statements included *completed all tasks* (repeated in 45 % of the retrospectives), *too much to do* (15 %), *efforts were not accurate* (35 %) and *good estimates* (30 %).

Regarding the need for clarifying instructions, the retrospective statements were as follows: *I got help when I needed it* (repeated in 15 % of the retrospectives), *incomplete specifications* (13 %) and *lack of information on how the system should work* (13 %). These discussions obviously concerned missing instructions for the development tasks and a lack of information on the requirements.

4.3.3 What Corrective Actions are Developed for the Recurring Discussions? (RQ 3.3)

A total of 19 % of all corrective actions were developed for the recurring discussions. Table 11 presents the developed corrective actions and the related recurring discussions (corrective actions were only developed for the negative experiences). The greatest number of corrective actions were developed for the discussions on *estimation accuracy* (8 % of all corrective actions) and the *state of bug fixing* (6 %). Despite the developed corrective actions, the statements on the recurring topics kept repeating themselves.

4.3.4 Reflections

Based on the results presented in this section, we conclude that the discussion topics varied over time and they also reflected the evolution of the development organisation. In a

Table 10 The recurring discussions and statements (categorised into process areas and topic types, and the percentage of retrospective meetings in which each statement occurred)

Discussion	Statement	Process Area	Topic Type	Recurrence %
The state of bug fixing				
	1) A lot of bugs	Implementation work	Task outcome	18 %
	2) Low bug count	Implementation work	Task outcome	15 %
	3) Fixed a lot of bugs	Implementation work	Task progress	38 %
Estimation accuracy and task completion				
	4) Efforts were not accurate	Implementation work	Task estimations	35 %
	5) Good estimates	Implementation work	Task estimations	30 %
	6) Too much to do	Implementation work	Resources & scheduling	15 %
	7) Completed all tasks	Implementation work	Task progress	45 %
The need for clarifying instructions				
	8) Incomplete specifications	Sales and requirements	Task outcome	13 %
	9) Lack of information on how the system should work	Implementation work	Instructions	13 %
	10) I got help when I needed it	Implementation work	Cooperation	15 %

longitudinal analysis, stage 1 represents a small software organisation with only one development team. The organisation adopted Scrum at the beginning of the observation period. The retrospective discussions focused on *instructions*, *learning*, *cooperation* and *tools* — logical topics for an incipient software organisation. During stage 2, the organisation consisted of two

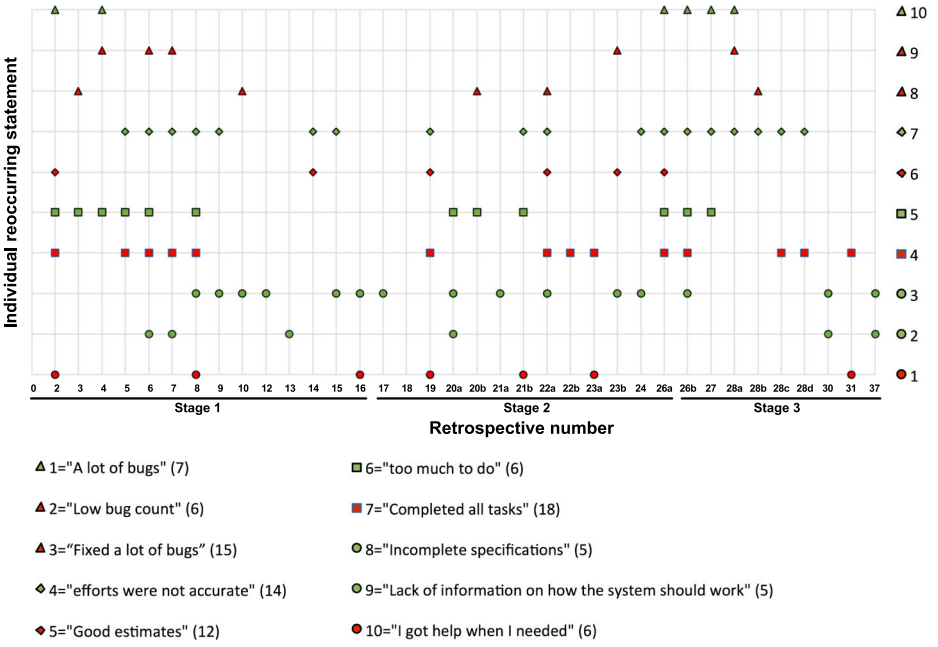

Fig. 8 Recurrence of similar retrospective statements over the retrospective timeline. *The y-axis lists the recurring statements numbered 1–10, colour coded as follows*: *negative = red and positive = green. The total number of recurrences are in parentheses*. The shape of the marker groups together statements from the same discussion

Table 11 Corrective actions for dealing with recurring negative experiences (*share = the share of corrective actions out of a total of 180 corrective actions*)

Discussion[1]	Statement	Corrective actions (retrospective number, see Table 1 and Fig. 8)	Share
Estimation accuracy	Inaccurate efforts	- *One way to help estimation could be WBD*[2] (*R5*), - *Add time/tasks for design, then implementation will be more straightforward* (*R5*), - *Stick to the time-boxed values and do not accept more stories* (*R19*), - *Average story points from 10 sprints* (*R19*), - *Estimate bigger numbers and/or accept less* (*R19*), - *Kanban-style dev* (*R22*), - *Budget time for doing prototypes* (*R23*), - *Familiarising (developers) with the stories beforehand* (*R23*), - *Factor the uncertainty factor into the time estimation* (*R23*), - *More clear requirements* (*R23*), - *Monitor the time that you spend and learn from your own estimates* (*R23*), - *When getting a new large story, allocate design and analysis story for the sprint* (*R23*), - *Make dedicated investigation task for features planned in next sprint* (*R23*), - *Estimation should be added on getting familiar with the project/component* (*R26b*), - *Those who know about the area of the story should tell about the work that is needed* (*R31*).	8 %
	Too much to do	- *Include realistic amount of work in the sprint and leave some time to settle quality debt* (*R2*), - *Use previous sprint as a reference of how much story points we can handle* (*R14*).	1 %
State of bug fixing	A lot of bugs	- *Enforce check-style on commit* (*R2*), - *Enforce the check-style checks before commit: add the plugin to the development tool* (*R2*), - *Reserve more time for testing when implementing new features* (*R21b*), - *Fixing debt about PARAM < component > verification* (*R21b*), - *Developer should reserve more time for testing and be more careful with it* (*R21b*), - *Remember to test all X methods/APIs that are related to your changes* (*R21b*), - *Test, test, test..* (*R31*), - *Test the issue in the actual application at least once* (*R31*), - *Testing in different environment: make testing in local environment and then in dev-test* (*R31*), - *More testing: the bigger the story is the more testing is required* (*R31*).	6 %
Clarifying instructions	Incomplete specifications	- *Use cases: a better understanding what is asked and what is needed to be achieved* (*R28b*), - *Create UML diagrams from source code to pin-point problems and design* (*R28b*).	1 %
	Information on the system	- *Architecture spec is missing and needs to be made* (*R4*), - *Add system overview descriptions/depictions to intranet* (*R4*), - *Add stories to current Sprint to add missing design document* (*R6*), - *Add JAVADOC tags wherever they are missing* (*R6*), - *Fast diagrams: draw diagrams on whiteboard, take picture and attach it to story* (*R6*).	3 %

[1] See section 4.3.2 for a description of the specific recurring discussions

[2] WBD refers to the Wideband Delphi estimation method

software development teams and faced the challenge of rapid growth during the second year. In comparison to stage 1, the retrospective discussions emphasised *task outcome* and *task progress*. In terms of the task outcome, the team members started to discuss the increasing number of software defects; they often mentioned the problem of 'a lot of bugs'. They also repeated the negative statement on task progress: 'too much to do'. Furthermore, stage 2 discussions rarely referred to positive experiences regarding task completion or progress. These problems eventually drove the organisation to lengthen the sprints to 3 weeks. During stage 3, the organisation had six teams and the duration of the sprints was further lengthened to 4 weeks by adding one additional week for testing. During that time, the recurring discussions on the high amount of bug fixing and open defects ended. During stage 3, the retrospective topics included somewhat similar topic types as in stage 2, but *cooperation* became a common topic type once again. Positive experiences with software testing also became more common. Additionally, the team members kept repeating the following points: 'I got help when I needed it' and I 'completed all tasks'. The statements reflect positive software engineering experiences important for the successful development work.

Our analysis revealed that certain discussions recurred over a long period of time. In the context of lean development, the problem of repeatedly regenerating the same list of retrospective findings is also recognised (Poppendieck and Poppendieck 2007). The recurring discussions in this case had to do with estimation accuracy, the state of bug fixing and the need for clarifying instructions. We hypothesise that the problems behind these three repeating discussions are different in nature.

We call the first recurring problem type 'trivial scapegoat' problem. For example, strive for better specifications and information about the system was such a problem. Here, a rather simple solution is desired as a solution to a complex problem. This discussion kept recurring because a simple problem was used to conceal the more complex phenomena behind it, and the root causes could not be solved without tackling the complex phenomena. In this case, the actual problem was related to poor communication and understanding of the system features. This problem arose from multiple levels, starting with the need to gather the required information from customers and understanding it correctly at the product-owner level. Furthermore, the communication problems between product owners and development teams increased the overall lack of required knowledge among the development teams. For this problem, a trivial scapegoat was presented in the retrospectives by stating that the specifications are poor or lacking. This problem kept recurring because the underlying challenges were much more complex than simply improving the specifications and documentation involving, for example, working practices and resourcing.

The second type of recurring problem is the 'unsolvable problem'. The estimation accuracy discussion was a combination of the previously described trivial scapegoat problem and unsolvable problem. Poor estimation accuracy is a complex problem that includes many uncontrollable factors on the team level. The whole concept of estimations was somewhat unclear for the team members. They thought that development would be much easier if the estimates were more accurate. However, this seems to be just an easy way of concealing a complex problem, such as a lack of communication, resourcing challenges and scheduling problems; thus, participants used estimation accuracy as a trivial scapegoat for a larger problem. In addition, the estimation accuracy seemed also to be an unsolvable problem. The inherent uncertainty of task estimations and constant changes in the estimated tasks cause inaccuracy. Considering all the uncertainties of software development and the organisational communication challenges, it is unrealistic to assume that it would be possible for the developers or product owners to be able to achieve

highly accurate task estimates. The high number of developed corrective actions without any significant effect during the course of 3 years corroborates the fact that the estimation problems were extremely difficult for the team members to solve.

The third type of problem leading to recurring retrospective discussions was a 'naturally recurring problem'. An example of this type of problem is the high number of bugs. In software development, bugs occur naturally and constantly, and the development team is well in control of temporarily improving this problem and decreasing the number of open bugs by investing in bug fixing and early testing activities. However, the problem kept recurring since the teams were incapable of solving the real causes due to external factors, including overly tight schedule pressures. The related discussions lasted for approximately 2 years. After the organisation invested enough effort in software testing and bug fixing, the problem stopped recurring in the discussions.

A fifth of all corrective actions were developed as a result of the recurring discussions (see Table 11). The team members developed the highest number of corrective actions for the estimation problems, and despite the repeating discussions and numerous corrective actions, the estimation accuracy did not improve. The team members also developed a high number of corrective actions for the software quality problems and, in contrast to the estimation problems, these actions had an effect and the problems decreased during the observation period. As a conclusion for our analysis of how the retrospective discussions evolved over time, we state that it is useful to identify the types of recurring discussions in the retrospective meetings. The recurrence could be a sign of unproductive discussions and ineffective corrective action innovations in the cases of trivial scapegoat problems or unsolvable problems. In the case of naturally recurring problems, identifying such a phenomenon could help target the analysis towards issues that cause the recurrences, not just the causes and corrective actions for the problem itself.

4.4 How Well do the Retrospective Discussions Correspond to the Development Repository Data? (RQ 4)

We studied how well the retrospective discussions related to the actual development status in the organisation. We analysed the recurring retrospective discussions and compared the changes in discussion topics to the task backlog system and the bug repository (see section 3.4). We focused our analysis on two recurring discussions, *estimation accuracy* and *the state of bug fixing*, each of which was described in section 4.3.2. These discussions were selected based on the availability of repository data that could be compared with the discussions.

4.4.1 Estimation Accuracy

Figure 9 presents the level of task estimation accuracy in the development sprints and maps it together with the outcomes from the team-level retrospectives. The data cover the timeline from the 20th retrospective (stage 2) to the 37th retrospective (stage 3). The task estimate data were only available for stages 2 and 3, since the company did not record the estimates and actual efforts during stage 1. The results show that the retrospective statements are somewhat contradictory. During stage 2, statements made in the 20th and 21th sprint retrospectives focused on 'good estimates', whereas in the 22th and 23th sprint retrospectives the statements more reflected the opinion that 'efforts were not accurate'. The estimations were relatively accurate in 21th sprint and inaccurate in the 23th sprint, which are in line with the statements. However, the estimation accuracy is similar for the 20th and 22th sprints, whereas the retrospectives found the opposite

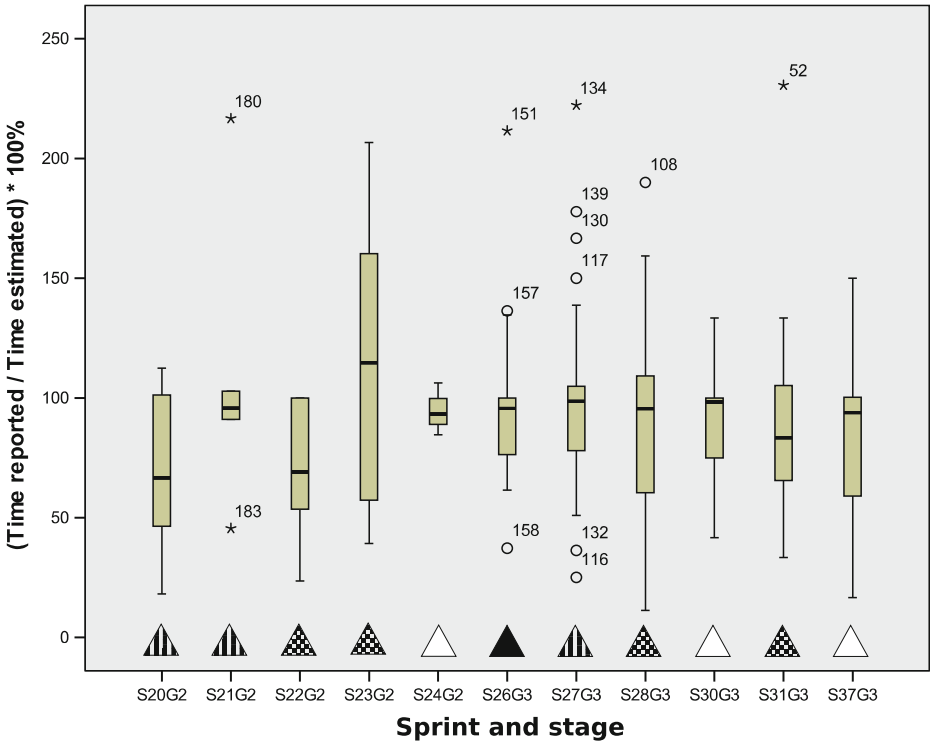

Fig. 9 Boxplot of the actual task estimation accuracy vs. the outcomes from retrospectives for the development sprints. *Notation on x-axis*: *SxxGy = Sprint number xx stage y. Triangles*: *striped* = '*good estimates*'; *checkerboard* = '*efforts were not accurate*'; *black = both statements were made*; *white = no statement regarding the estimates* (*the circles denote outliers and the stars extreme outliers*)

perceptions. The retrospectives did not reveal a high degree of measured estimation accuracy in the 24th sprints retrospective. Furthermore, in the 27th sprint retrospective, participants perceived 'good estimates' and in the 28th and 31th sprint retrospective that the 'efforts were not accurate'. These results are also contradictory due to the fact that the estimation accuracy in the 27th sprint was not clearly different than it was in the 28th and 31th sprints. Additionally, the retrospective for 26th sprint found that the estimations were both inaccurate and accurate. Obviously, the retrospective statements do not reflect the estimations for all tasks in a sprint, since the deviation in the estimation accuracy is high, as is visible in the numerous outliers in Fig. 9.

4.4.2 State of bug Fixing

Figure 10 maps the retrospective statements on the state of bug fixing into the timeline of open bugs (data from the bug repository). It seems that these statements are in line with the real-world status of open bugs. When the number of open bugs increased (in comparison to the past), the related retrospective found that 'a lot of bugs' were present. In contrast, when the number of open bugs decreased, the related retrospective found that the company had 'fixed a lot of bugs'. Furthermore, the statement 'low bug count' occurred when the number of open bugs was approximately 90 or less. The teams did not demonstrate the actual bug counts before the retrospectives. The figure also divides the timeline into stages 1–3. It seems that in

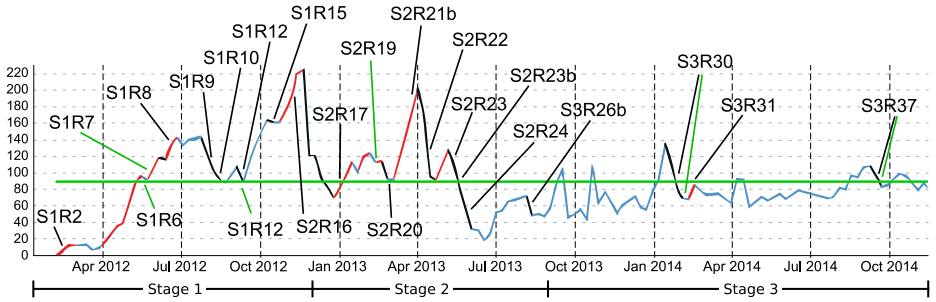

Fig. 10 Line chart showing the timeline of bug counts in the task repository and related retrospective discussions. *Blue curve = the number of open defects*; *red = an increasing number of defects were discussed in the retrospectives*; *black = a decreasing number of defects were discussed in the retrospectives*; *green label pointers = that a 'low bug count' was mentioned in the retrospectives. Labels*: *S < x > R < n > = stage x, retrospective number n*

stage 3, the number of open bugs stabilised at less than 90 open bugs, which could explain why the number of statements on the high number of bugs also decreased.

4.4.3 Reflections

We analysed the correspondence between the retrospective discussions and the repository data from the task repositories. This analysis revealed that certain discussions, such as remarks on the high or low number of bugs, rather accurately reflected the state of development depicted by the repository data. Other discussions, such as the comments on poor estimation accuracy, did not match the situation reflected by the task repository data on task estimates and actual efforts. In our analysis, we identified several phenomena related to these two recurring discussions.

Our first finding was that these recurring discussions were related to different types of problems and that the reasons for the recurrences were different. In the case of the discussions on the high number of open bugs, the problem was easy for the team members to recognise and they were able, at a team level, to react to the problem. The problem kept recurring and the related discussions lasted for approximately 2 years matching the measured open bug trends. The team members also developed a high number of corrective actions for the software quality problems. In comparison to the estimation problems, these problems did reflect reality and the problem did decrease during the observation period, when the number of open bugs levelled off at below 90 (see Fig. 10). However, we cannot explain why participants felt that less than 90 open bugs was a low bug count.

The other recurring discussion regarding estimation accuracy concerned a problem that was different in nature. The problem was extremely difficult, if not impossible, for the team to solve. These difficulties with the estimation concept caused team members to use the estimation accuracy as an excuse, a trivial scapegoat. This was visible in the way that participants hoped the better estimates would solve such problems as a lack of communication between the team and product owners. The retrospective discussions on estimation accuracy were contradictory in comparison with the estimation accuracy data from the task repository. In addition, despite the high number of repeating discussions and corrective actions for the estimation problems, the estimation accuracy did not improve (see Fig. 9).

Furthermore, the repeating discussions on the need for clarifying instructions were also a problem, with complex uncontrollable factors being easy to blame in the case of failure. Even though we did not have triangulating data to further explain this finding, our earlier study

revealed that the case organisation struggled with the collaboration problem with respect to stakeholders (Lehtinen et al. 2015b). It resulted in a lack of information exchange between sales & requirements, the product owners and developers. The underlying problem was related to the complexity of the business domain, including the high number of customers and the fact that third parties were difficult to control and collaborate with.

The outcome of an individual retrospective provided only a narrow sample of the problems and corrective actions regarding the software development practice. When the retrospective outcomes were combined over a longer period of time, they collectively provided both a more detailed and broader view of the perceived issues and relationships interconnecting the planning work, implementation work and software testing process areas (see Fig. 6). Such a collective software engineering knowledge base has been presented as a valuable asset (Anbari et al. 2008). Due to the specific temporal context and human factors affecting the outcome of each individual retrospective, the combined overview does not accurately illustrate the current situation. The past was not equal to the present. For example, the combined overview did show that there had been a problem with the high number of bugs. However, it did not show that the problem decreased over time. Additionally, the combined overview showed that the teams have problems with estimations. However, the deviations in the overall estimation accuracy (see Fig. 9) reveal that significant differences between the stated 'accurate' and 'inaccurate' development sprints in fact barely existed.

5 Implications and Evaluation of the Research

In this section, we discuss the implications of the results of this study for team-level retrospective practice. In addition, we evaluate the validity of this research study and discuss potential validity threats.

5.1 Implications for Continuous Retrospective Practice

Based on the results and discussion presented in this paper, we identified several phenomena that could affect the outcome and efficiency of the team-level retrospectives. Steering the focus and development of process improvement proposals in the retrospectives in the direction of controllable and valid problems would be valuable with respect to preventing the team members from producing retrospective waste by discussing and trying to solve invalid, overly complex or uncontrollable problems.

We identified four central viewpoints that can be used to provide guidance for the practical implementation of team-level retrospectives. These viewpoints work both as a set of heuristics for retrospective facilitators and as initial hypotheses for further empirical research. Table 12 summarises these viewpoints.

The first viewpoint is *bias*. One of the problems with the case organisation's retrospective outcome had to do with the participants' biased understanding of the current development situation. The outcomes of the case organisation's retrospectives did not always correspond to the evidence from the development repositories. The reliability problem of learning from experience has been identified in earlier studies. For instance, Jørgensen and Sjøberg (2000) have described different biases that affect how people apply their experience when learning from past events. They argue that without knowledge of and reflection on the potential biases, experience-based knowledge will include much incorrect information and invalid causal relationships. They suggest that the

Table 12 Summary of the study implications and initial hypotheses

Viewpoint	Hypothesis
Bias	Lack of individual knowledge and biases in interpreting personal experiences correctly leads to incorrect experience-based information in retrospective discussions and invalid inferences regarding causal mechanisms if participants are not aware of the potential biases and fact-based evidence is not used to support the analysis.
Controllability	The team-level retrospectives tend to focus on topics that the development team can control, and the team's improvement proposals for issues beyond the control of the team tend to remain ineffective. Teams require support in resolving issues that are beyond their control.
Complexity	Some problems are too complex to be solved through team-level retrospective practice, and a team's improvement suggestions for such complex problems tend to remain ineffective. Teams require support in resolving issues that are highly complex.
Recurrence	In continuous retrospectives, certain discussions occur repeatedly. Repeating discussions are potential signs of waste in the retrospective practice due to issues of controllability, complexity or a natural recurrence of the discussed problems. It is useful to identify the recurring discussions and guide the retrospectives to avoid repeating the unproductive discussions.

experience providers would need to consider alternative perspectives and critique their own judgments, and in addition, they should use training and increase awareness about the limitations and biases in experience-based learning. One approach to improving the validity of retrospective analysis is to introduce fact-based evidence to the retrospective in order to base the analysis on a more objective picture of the real situation instead of the potentially biased personal experiences of the participants. Bjarnason et al. (2014) have proposed using evidence-based timelines that provide participants with a visual timeline of project events with factual evidence that serves as a memory prompt in discussions. These timelines should be constructed before the retrospective meeting from various sources. In our case, the defect data and the estimation accuracy measures could have been good examples of evidence to present in the timelines to support retrospective discussions. In this case, the evidence was not used, even though the bug data in particular would have been readily available.

The second viewpoint is *controllability*. We recognised that the team-level retrospectives resulted in corrective actions that the team members were able to control (eg the work practices of the team). We also recognised that part of the retrospective outcome could not be controlled at the team level, but that it could be controlled at the organisation level (eg processes and testing resources). Resolving such findings requires organisational support. Furthermore, part of the retrospective outcome was external to the whole software organisation (eg customers & users) and requires extensive collaboration both within and outside of the company.

The third viewpoint is *complexity*. Solving a problem through retrospective practice requires controlling its causes. Some problems are caused by a complex network of causes that are too complicated to be correctly analysed and resolved within the context of a team-level retrospective. Such problems have the potential for causing waste in terms of valuable time and effort on the part of the team in the retrospective meetings. If the problem is too complicated to solve, the discussions tend to repeat themselves, and ineffective corrective actions accumulate over time without any real effect. Therefore, in order to prevent waste it would be beneficial to recognise such complex problems and find better methods and different forums for addressing them.

The fourth viewpoint is *recurrence*. Our analysis identified different types of recurring discussions in the retrospective meetings. The software organisation should recognise the retrospective

discussions that keep recurring. These discussions could reflect a problem requiring organisational support (or external support). The discussions might also reveal trivial scapegoat problems. In any case, the repeating of such problems indicates potential waste in the retrospective practice.

5.2 Evaluation of the Research

This section discusses the validity of our empirical results using a validation scheme presented by Runeson and Höst (2009). We will present the construct validity in section 5.2.1 and the external validity in section 5.2.2 and focus on the reliability of the study in section 5.2.3.

5.2.1 Construct Validity

Construct validity concerns how accurately the applied operational measures truly represent the concepts that researchers are trying to study (Runeson and Höst 2009). In this study, these included the retrospective outcomes and the task backlog.

Our results are limited to the documented outcomes, which do not capture all the details covered in the face-to-face retrospectives. We were not able to confirm the detailed face-to-face discussions. The researchers were not present in the team-level retrospectives and the discussions were not tape-recorded. There is a threat that the raw data do not accurately represent all the discussions that took place in the retrospective meetings and it is likely that the practitioners did not document all the discussions that took place during the meetings. The documented outcomes of the retrospectives included the cause-effect diagrams, together with the voting results and the related corrective actions. Due to the detailed retrospective statements, we feel that such raw data made it feasible to synthesise the retrospective discussion topics and compare the change in topics over time. This validity threat affects the total number of retrospective statements; however, it is unlikely that there would be missing categories at the level of process areas or topic types since the teams documented the discussions by collaboratively using the software tool during the discussions. In other words, the documentation process is not dependent on the recollections or interpretations of a single individual.

The teams used a specific software tool to assist them in collaboratively collecting and documenting the retrospective outcomes in diagram form. The tool probably made it easy for the participants to record all the relevant statements, but we do not know how the tool affected the retrospective meetings. In our earlier work (Lehtinen et al. 2015a), we compared the use of diagrams and structural lists in retrospectives, and those results indicate that diagrams might well increase the number of findings and relationships in the outcomes and help in organising the findings. Diagrams were also perceived to be visually more attractive.

The use of the categorisation system might also bias the results. It dissipates details and emphasises similarity, and thus our analysis of the discussion topics does not represent the full discussion content. This threat applies to our analysis of the process-area and topic-type categories. In our analysis of the recurring discussions and their correspondence to the repository data, we analysed the detailed retrospective statements. We used qualitative techniques to consolidate the results on the outcomes from the retrospectives, and we also modified the categorisation system to correspond with the raw data.

The facilitators of the team-level retrospectives varied. At least four different Scrum masters facilitated the retrospectives and led the discussions. Most of these facilitators were present at all retrospective stages (stage 1, stage 2 and stage 3). There is a threat that the facilitators personally affected the outcome of the retrospective meetings, but variation in the facilitators is unavoidable in a longitudinal industrial research setting.

The task backlog was used to investigate the retrospective discussions on bug counts and estimation accuracy in relation to the repository data. We compared the retrospective discussions and the actual estimation accuracy and the number of open bugs over time. There is a threat with respect to construct validity in terms of the reliability of the task backlog. It is possible that sometimes the realised effort was reported as being equal to the planned effort, which would bias the analysis in terms of the actual estimation accuracy. In most cases, the realised effort differed from the planned effort. Similarly, the actual state of open bugs was based on the number of open bugs reported.

5.2.2 External Validity

External validity is concerned with the generalisability of the findings (Runeson and Höst 2009). Our results are based on the findings from one distributed Scrum organisation only. Thus, we cannot generalise about the exact quantities, only about the theories and explanations describing the outcomes of the team-level retrospectives in similar case contexts.

We believe that the external validity of the discussion topics covered in the team-level retrospectives in similar case contexts is high. It is reasonable to claim that the team-level retrospectives consider the experiences close to the team members. It is also reasonable to claim that the outcomes reflect common software development challenges and successes, as presented in prior literature. Furthermore, the retrospective statements will likely keep recurring if the problems are not solved or if the experiences themselves keep recurring. In contrast, the detailed discussions and distributions of the retrospective statements across the topic types and process areas likely vary in different cases. It is also possible that new case contexts could extend the discussion topics already recognised. As we concluded in our prior study (Lehtinen et al. 2014a), the case contexts vary and therefore a case-specific analysis is likely needed every time a failure occurs.

5.2.3 Reliability

Reliability considers the extent to which the data and analysis are dependent upon the researcher (Runeson and Höst 2009). The level of reliability regarding the raw data is high due to the fact that it was provided by the case organisation. Therefore, the threats to reliability are mostly related to data analysis and the ultimate conclusions drawn as a result.

The study results are based on a qualitative categorisation (coding) of the retrospective statement data by the researchers. The use of a modified categorisation system was not evaluated. However, we studied the inter-rater agreement of the original categorisation system in our prior study (Lehtinen et al. 2014a). Kappa values for the process-area and topic-type dimensions were 0.65 and 0.55, respectively, which we concluded were good and moderate agreements. It is possible that the new categories that emerged during this study are more researcher dependent.

The modified dimensions do not equal those used in the prior study, but the modifications were minor. The process area categorisation was extended by adding a few new categories. The categories used in both studies were sales & requirements, management, implementation work, testing and deployment. We found that the level of agreement for these areas remained at a similar level. The new, Scrum-specific process areas include sprint planning, product owner and Scrum master. Similarly, the changes in the topic-type categories were minor. Most of the types remained the same and only minor improvements were added; for instance, the prior study includes a topic type called instructions & experiences, whereas the modified version includes instructions and experience as two separate topic types.

The results for the recurring discussions could be validated by triangulating the repository data only regarding the discussions on the number of bugs and the estimations accuracy. No data were available to validate the recurring discussions concerning the instructions and specifications. This is a validity threat since triangulating data regarding the third type of discussion could reveal new findings. The lack of triangulating data does not invalidate the finding regarding the recurrence or frequency of the third discussion, but it does prevent us from drawing further conclusions based on the nature of the recurrences. Finally, the final results were not verified by the study participants; only preliminary findings were discussed with a company representative.

6 Conclusions and Future Work

This was a longitudinal case study on team-level retrospective outcomes that emphasises the change over time and the recurrence of the outcomes. The study makes five contributions. First, we synthesised and made generalisations about the discussion topics that occurred in the team-level retrospectives. We found that continuous retrospectives provide a detailed analysis of both the positive and negative experiences of the software development teams. However, the discussions in the team-level retrospectives focused heavily on the areas close to the development team and topics that were directly related to the implementation work. The positive discussions were even more strongly focused on the implementation work.

Second, we analysed the links between the developed corrective actions and the retrospective discussions. We found that the corrective actions were commonly developed for certain topic types. These included *task outcome*, *work practices*, *task difficulty*, *instructions*, *estimations* and *resources & schedules*. We hypothesise that these topic types revealed problems, which were perceived as controllable by the team members. To ensure effective results, the team-level actions should be guided to target the controllable problems, but the team should simultaneously make sure that important problems are not discarded because of a lack of control at the team level.

The third contribution was our analysis of the changes in the retrospective discussions over time. We found that as the young, single-team organisation grew over time, the emphasis moved first even more heavily into implementation work and software testing and that discussions typically focused on the topics of task progress and task outcome. Later, as the organisation grew more, the discussion moved from implementation work to a wider coverage of the process areas, including resources and scheduling, the product owner and cooperation. We conclude that the retrospective discussions do change over time and reflect the evolution of the organisation over time, which should be taken into account when analysing retrospective data over long time periods.

Fourth, based on our analysis we found that recurring discussions exist in continuous retrospectives and are a potential cause of waste in retrospective practice. We

recognised three repeating discussions in the case organisation: *estimation accuracy and task completion*, *the state of bug fixing* and *the need for clarifying instructions*. We identified that the nature of the problems leading to these repeating discussions were different. The problems included complex problems that are practically unsolvable at the team level, naturally recurring problems and 'trivial scapegoat' problems, ie problems that are easy to identify and difficult to solve. Our analysis of recurring discussions also revealed that the reliability of the retrospective outcome varies, and incorrect interpretations and discussions exist. A large number of corrective actions were developed for the recurring discussions. The issue of recurrence should be addressed and potential unproductive discussions handled as part of the retrospective practice.

Fifth, we propose four viewpoints, which could be useful for evaluating the retrospective outcomes. These viewpoints are *bias*, *controllability*, *complexity* and *recurrence*. The reliability of the problems presented in the retrospectives should be verified via evidence in order to avoid repeating the same discussions based on biased interpretations or incorrect understandings. Controllability and the complexity of the problems are important to consider in order to select appropriate actions for resolving them. Furthermore, it would be reasonable to analyse the recurring retrospective findings to spot potential waste and detrimental effects in terms of repeating, but unproductive discussions.

We call for future work and replicating studies to increase the external validity and reliability of our results. It would be important to empirically study and validate our initial viewpoints and hypotheses on how to use our findings to guide continuous retrospective practice. In addition, comparative research on team-level and organisation-level retrospectives would be beneficial for understanding how to facilitate the development of corrective actions and implement them at the correct level of organisational control.

References

Anbari FT, Carayannis EG, Voetsch RJ (2008) Post-project reviews as a key project management competence. Technovation 28:633–643

Bjarnason E, Hess A, Berntsson Svensson R et al (2014) Reflecting on evidence-based timelines. IEEE Softw 31(4):37–43

Bjørnson FO, Wang AI, Arisholm E (2009) Improving the effectiveness of root cause analysis in post mortem analysis: a controlled experiment. Inf Softw Technol 51(1):150–161

Card DN (1998) Learning from our mistakes with defect causal analysis. IEEE Softw 15(1):56–63

Cerpa N, Bardeen M, Kitchenham B et al (2010) Evaluating logistic regression models to estimate software project outcomes. Inf Softw Technol 52(9):934–944

Cockburn A (2002) Agile software development. Addison-Wesley

Collier B, DeMarco T, Fearey P (1996) A defined process for project post mortem review. IEEE Softw 13(4):65–72

Derby E, Larsen D, Schwaber K (2006) Agile retrospectives: making good teams great. Pragmatic Bookshelf Raleigh, NC

Dingsøyr T (2005) Postmortem reviews: purpose and approaches in software engineering. Inf Softw Technol 47(5):293–303

Dingsøyr T, Moe NB, Nytrø Ø (2001) Augmenting experience reports with lightweight postmortem reviews. In: Anonymous PROFES '01 Proceedings of the Third International Conference on Product Focused Software Process Improvement, p 167-181

Drew Procaccino J, Verner JM, Overmyer SP et al (2002) Case study: factors for early prediction of software development success. Inf Softw Technol 44(1):53–62
Egorova E, Torchiano M, Morisio M (2010) Actual vs. perceived effect of software engineering practices in the Italian industry. JSS 83:1907–1916
Glass RL (2002) Project retrospectives, and why they never happen. IEEE Softw 19(5):111–112
Jalote P, Agrawal N (2005) Using defect analysis feedback for improving quality and productivity in iterative software development. In: Anonymous proceedings of the information Science and communications technology (ICICT 2005) Infosys technologies limited Electronics City, Hosur Road, Bangalore, India, p 701 - 714
Jin ZX, Hajdukiewicz J, Ho G et al (2007) Using root cause data analysis for requirements and knowledge elicitation. In: Anonymous international conference on engineering psychology and cognitive ergonomics (HCII 2007). Germany Springer Verlag, Berlin, pp 79–88
Jones C (2008) Software tracking: the last defense against failure. Crosstalk J Def Softw Eng 21
Jørgensen M, Sjøberg D (2000) The importance of NOT learning from experience. In: Anonymous Proc. Of European Softw. Process Improvement, EuroSPI'2000, p 2.2-2.8
Kaur R, Sengupta J (2011) Software process models and analysis on failure of software development projects. Int J Sci Eng Res 2(2):2–3
Keil M, Cule PE, Lyytinen K et al (1998) A framework for identifying software project risks. Commun ACM 41(11):76–83
Lehtinen TOA (2014) Development and evaluation of a lightweight root cause analysis method in software project retrospectives, Aalto University
Lehtinen TOA, Mäntylä MV, Vanhanen J (2011) Development and evaluation of a lightweight root cause analysis method (ARCA method) – Field studies at four software companies. Inf Softw Technol 53(10):1045–1061
Lehtinen TOA, Mäntylä MV, Vanhanen J et al (2014a) Perceived causes of software project failures – an analysis of their relationships. Inf Softw Technol 56(6):623–643
Lehtinen TOA, Virtanen R, Viljanen JO et al (2014b) A tool Supporting root cause analysis for synchronous retrospectives in distributed software teams. Inf Softw Technol 56(4):408–437
Lehtinen TOA, Mäntylä MV, Itkonen J et al (2015a) Diagrams or structural lists in software project retrospectives – an experimental comparison. J Syst Softw 103(May):17–35
Lehtinen TOA, Virtanen R, Heikkilä VT et al (2015b) Why the development outcome does not meet the product owners' expectations?. In: Anonymous Agile Processes, in Software Engineering, and Extreme Programming Springer, p 93-104
McLeod L, MacDonell SG (2011) Factors that affect software systems development project outcomes: a survey of research. ACM Comput Surv 43(24):24–55
Moløkken-Østvold K, Jørgensen M (2005) A comparison of software project overruns - flexible versus sequential development models. IEEE Trans Softw Eng 31(9):754–766
Poppendieck M, Poppendieck T (2007) Implementing lean software development: from concept to cash. Pearson Education
Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164
Runeson P, Höst M, Rainer A et al (2012) Case study research in software engineering: guidelines and examples. John Wiley & Sons, Hoboken, NJ, USA
Schwaber K, Sutherland J (2011) Scrum guide. Scrum Alliance
Stålhane T (2004) Root cause analysis and gap analysis - a tale of Two methods. In: anonymous EuroSPI 2004, Trondheim, Norway, vol 3281. Springer Verlag, Berlin Heidelberg, pp 150–160
Stålhane T, Dingsøyr T, Hanssen G, et al (2003) Post mortem–an assessment of two approaches. Empirical Methods and Studies in Software Engineering :129-141
Tiedeman MJ (1990) Post-mortems - methodology and experiences. IEEE J Selected Areas Commun 8(2):176–180
Xiangnan L, Hong L, Weijie Y (2010) Analysis failure factors for small & medium software projects based on PLS method. In: Anonymous The 2nd IEEE International Conference on Information Management and Engineering (ICIME), p 676-680
Yin RK (ed) (1994) Case study research: design and methods, vol 2nd. SAGE Publications, United States of America

Timo O.A. Lehtinen is a science adviser at the Academy of Finland. He received D.Sc. in software engineering from Aalto University, Finland in 2015. He is a former postdoctoral researcher at Aalto University and his research work have focused on retrospective methodologies and outcome, which he has studied both in the software industry and education contexts. He has years of software project management experiences in industry.

Juha Itkonen works as a postdoctoral researcher at the Department of Computer Science, Aalto University, Finland. His research interests focus on experience-based and exploratory software testing and human issues in software engineering, including quality assurance in agile context. In his current research, he is working also on continuous software engineering practices and quality assurance in large-scale agile context. He conducts empirical research relying both on qualitative and quantitative methods and prefers research in industrial context. He received his D.Sc. degree in software engineering in 2012 from Aalto University.

Casper Lassenius is an associate professor at Aalto University. His current research interests include agile and lean software development, global software engineering, and software quality assurance. He has a D.Sc. degree from Aalto University.