Supporting Reflection in Software Development with Everyday Working Tools

Birgit Krogstie and Monica Divitini

Abstract Through their day-to-day usage collaboration tools collect data on the work process. These data can be used to aid participants' retrospective reflection on the process. The paper shows how this can be done in software development project work. Through a case study we demonstrate how retrospective reflection was conducted by use of an industry approach to project retrospectives combined with the examination of historical data in Trac, an issue tracker. The data helped the team reconstruct the project trajectory by aiding the recall of significant events, leading to a shift in the team's perspective on the project. The success of the tool-aided retrospective reflection is attributed to its organization as well as the type of historical data examined through the tool and the tool features for navigating the data. These insights can be used to help project teams determine the potential of their tools to aid retrospective reflection.

Introduction

Software development (SD) is highly cooperative work which has received considerable attention in CSCW [1–3]. In SD, mistakes are common as well as costly [4], and learning from experience is essential [5].

In SD organizations using large integrated systems to support and coordinate work, efforts to have the organization learn from experience may be integrated into predefined work processes and supported by tool functionality for collecting and providing experience data. The idea of experience factories [6] reflects a similar idea. In other settings, as in agile software development [7], learning from experience may be more informal and deferred to face-to-face project meetings and retrospective

B. Krogstie (✉) and M. Divitini
Norwegian University of Science and Technology, Sem Sælands vei 7-9, NO-7491, Trondheim, Norway
e-mail: birgitkr@idi.ntnu.no; divitini@idi.ntnu.no

M. Lewkowicz et al. (eds.), *Proceedings of COOP 2010*, Computer Supported Cooperative Work, DOI 10.1007/978-1-84996-211-7_9,

reflection sessions. *Project retrospectives* [8, 9] in agile SD are an example of what is known in the project management literature as project debriefings [10], which cover various methods for recording experiences from projects. Project retrospectives are conducted at the end of project phases or when projects are terminated. The major resources are participants' memory and collaborative effort to reconstruct the project, and various facilitation techniques exist (e.g. [11]). In this paper we generically refer to *retrospective reflection* to indicate activities conducted at the end of a project phase to rethink the working process with the goal of learning from experience. *Retrospective reflection* can be seen as an aspect of reflective practice [12] comprising reflection-on-action in context of the project.

One challenge, found to be among the reasons why many organizations choose *not* to conduct retrospective reflection [13], is the lack of adequate data to help participants reconstruct the process. Human memory is fallible in recalling a process of weeks or months. For other data sources to be useful in practice, however, they should provide access to data with an appropriate level of detail and enough context to help avoid oversimplification and examination of unimportant details [13].

The objective of this paper is to investigate the potential to support project teams' retrospective reflection by the use of historical data in *lightweight collaboration tools* [14]. These are tools typically providing basic functionality for collaborative activity without imposing much structure on the process. This allows flexible incorporation of the tool into the specific process. Taking lightweight collaboration tools into use requires little resources (e.g. acquisition cost, time for training) on part of users and organizations. Lightweight tools are in use in a large number of SD projects, some of which have very limited resources for organizing retrospective reflection sessions. Lightweight tools may be the choice of small organizations, teams doing cross-organizational work, teams basing their selection of tools and methodologies on local needs and preferences, or teams in organizations running 'lean' processes. As a side-effect of day-to-day work, the lightweight collaboration tools collect data originating in the work process. Such historical data may have the appropriate detail and context to aid participants' retrospective reconstruction of the process.

The CSCW literature has addressed the use of lightweight cooperation technology to support day-to-day project work, e.g. in software development [15–17]. Also, there is research on the use of data captured in collaboration tools to gain insights about collaborative work activity, as will be elaborated in the Background section. The potential to aid retrospective reflection on an entire project process by use of historical data in the various lightweight collaboration tools supporting that process was identified in [18].

For a project team to be able to determine the potential of their collaboration tools to aid retrospective reflection, they need knowledge about *what it is that makes a tool useful* for such a purpose. Such knowledge is also of value to collaboration tool designers who might want to extend the area of application of their tools to retrospective reflection.

Research contributions in this area should include empirical investigation of tool use as it unfolds in the context of retrospective reflection. This paper provides such a contribution by going into detail on how retrospective reflection is aided by the use of historical data in a collaboration tool in a SD project followed in an in-depth case study.

In the background section we outline relevant theory and related work from the CSCW field. The case section describes the SD project of our study and the project management tool used in that project. The research method section includes a description of the organization of the team's reflection workshop. Findings from the workshop are next described, followed by a section on discussion and implications, a section outlining a set of guiding questions regarding the potential of collaboration tools to support reflection in SD work, and finally our conclusion.

Background

In this section we briefly provide a background on distributed cognition as a way of understanding SD work and on the reconstruction of the process trajectory as important to retrospective reflection. We also address how the day-to-day use of collaboration technology results in the creation and storage of historical data.

In the analysis we have used distributed cognition [19, 20] to theoretically frame the problem. Distributed cognition is a perspective which has been used to understand learning [21], software development [22, 23], and cooperative work more generally [24]. Cooperative work can be seen as distributed across the people involved in the activity, through coordination of the processes between internal and external representations, and through time in such a way that results of earlier activity can transform later activity. Individual participants' mental representations [25] of the project process mediate day-to-day project work as well as reflection on the work. Strauss [26] argues that when people make sense of a process, they think of it as a *trajectory*. On this basis, retrospective reflection can be seen to involve the individual and collective reconstruction of the process trajectory, which includes identifying its significant events. The trajectory can have an internal or external representation (e.g. a diagram or a textual description). In sum, distributed cognition is a particularly useful framework for analyzing and designing for retrospective reflection because it helps shed light on the role of representations that mediate the reflective process, being used and transformed by participants individually and collaboratively.

In the day-to-day project activity shaping the project trajectory, cooperative work entails coordination, access to artifacts in a shared workspace, and formal as well as informal communication – aspects of work typically supported by cooperation technology. Formal communication, for which archiving is important, may for instance be conducted over email [27] and informal communication, e.g. about ongoing work, by synchronous [16, 28] and asynchronous chat. A shared workspace

to hold artifacts under development must be provided, as well as ways of providing co-workers with workspace awareness [29–31], including social awareness [32]. Coordination of complex work, e.g. its planning, monitoring and re-planning, requires coordination artifacts [1, 33], computerized tools ranging from simple to-do-lists and bulletin boards to advanced enterprise solutions. Among the tools used to aid the coordination of work are issue trackers [34, 35] used in Software Development.

As a consequence of the activity along the trajectory, data get stored. Depending on the usage of each tool, its data will reflect different aspects of the SD work, e.g. development and bug-fixing, stakeholder collaboration [36], and team-internal, informal communication [37, 38]. From a distributed cognition perspective, the data are external representations originating in there-and-then distributed cognition. The data may have the form of logs (events, communication), versions of project artifacts (e.g. code, documentation), and plans and status reports. Data are generally recorded in repositories that are accessible through the tools. The idea of capturing information relevant to a work process from tools and artifacts already used to support that work process has been explored in several ways in the context of SD. Some approaches involve the deliberate tagging of information relevant to others [39, 40]. The use of logs from a code management system to support coordination in a SD team has been proposed [41]: An event notification system and a tickertape tool for CVS messages was integrated with the code management system, improving communication in the team. The project memory tool Hipikat is designed to help new members of a distributed SD team learn from more experienced colleagues [42]. The memory contains the artifacts produced during development and is built by the tool with little or no changes to the existing work practices. The SEURAT system [43] is designed to capture design rationale in software development and was originally intended to support software maintenance. The system can also trace requirements and decisions [44] in the project. Aranda and Venolia [45] found that for purposes of coordinating SD work, data repositories from the development process (bug databases and email repositories) could provide some information about the bug fixing ‘stories’, as reconstructed by the researchers having access to all the repository data, but the data in the repositories were incomplete and sometimes erroneous. The study showed that the reconstruction of the stories, including the social, organizational and technical knowledge, required the participants in the bug fixing process to account for the stories. Aranda and Venolia also found that among the most problematic types of missing data in the repositories were missing links from the bug records to the source code involved.

In [18] it was argued that data collected in various collaboration tools in daily use in a project may be systematically utilized to aid retrospective reflection in the project team. From a distributed cognition perspective, the historical data in the tools can be considered external representations of the project process. Together with team members’ internal representations of the project process and external representations constructed in the team’s collaborative reflection effort, the historical data in the tools are a potential resource in the reconstruction of, and reflection on, the project process. The outcome of the reflection (i.e. lessons learned, manifest

in internal and external representations) subsequently informs the work practice, including the use of the collaboration tools from which the historical data were drawn. In the paper, a lightweight project management tool in the form of an issue tracker (Trac [46]) was used as an example of a tool with a potential to support retrospective reflection.

The aim of this paper is to demonstrate in detail how retrospective reflection can be aided by the use of historical data in Trac. Outlining how the historical data are utilized as an integral part of a systematic retrospective reflection approach with individual and collective steps, we stress how the collaboration tool plays a role in *knowledge building* as well as which *tool features* are particularly useful to support retrospective reflection in the team.

Case

Our case is a SD student team in an undergraduate, capstone project course. The projects have external customers, require a high level of independence in terms of choice of process, technology and solution, and are designed to provide a work setting as close to industry SD as possible. Meeting customer requirements is a primary goal, along with meeting the university's requirements for certain project deliveries, most notably a project report to be delivered in a preliminary, mid-term and final version. There are three to five students in the teams. The project takes up half the workload in the final (sixth) semester of a Bachelor of IT program.

As part of the course, a retrospective reflection workshop is arranged for each team at the end of the project based on a technique drawn from industry SD practice [8, 9]. With this technique, participants individually and collectively construct a timeline of project events. Opinions and feelings about what is positive and negative are added, e.g. by use of post-it notes in different colours or by use of curves indicating the 'ups and downs' of the process. The final steps of the workshop are directed at determining future action based on the analyzed experience. In the workshop of our course, we use the following main steps: the participants draw a timeline of their project, adding events considered significant. First, timelines are drawn individually and next a shared one is collectively constructed. The emotional aspects of the process are addressed by having participants draw curves (that we call 'satisfaction curves') along the timeline, indicating their satisfaction with the project at different times. Next follows a discussion of issues related to lessons learned. It was shown in a study that the workshops can help the teams share knowledge and reach new insights about their projects [47].

The project of our study had five team members (Fig. 1). Their task was to develop an application allowing people to use their mobile phone to keep track of the geographical location of their friends and engage in textual chat with them. The required solution was technically advanced, including a server, an administration client and a mobile client and the use of services from a provider of geographical positioning data.

Fig. 1 The project team of the case study: snapshots from everyday project work

To support the management of their development work, the team chose to use an issue tracking tool called Trac (http://trac.edgewall.org/). Trac is a web application that supports the organization of work into tasks and phases with milestones, and is lightweight in the sense that it contains only the basic features necessary to manage SD work and requires little resources to be taken into use. Tasks are defined by *tickets* that are assigned to the users, e.g. team members. Trac also contains a wiki, allowing for any contents to be flexibly added. Trac provides a set of views into the files managed by an underlying file versioning system (SVN) containing e.g. source code and project documents. When a team member uploads (i.e. *commits*) new or modified files to the file management system, e.g. from the development environment in which source code is produced and tested, it can be seen in Trac as a new *changeset*. Usually, the user adds a comment on what the changeset is about. All versions of all files that have been committed to SVN, as well as the differences between file versions, can be viewed through Trac.

The Trac timeline (see Fig. 2, screenshot (a)) contains an item for each changeset (and its comment), wiki update, ticket update, and milestone completion. Each item has a date and time of the update; an identifier linking to the particular version of the project artifact(s) affected; and the name of the user doing the update. In Fig. 2a, for example, the comment associated with changeset [86] reflects an update of source code to the SVN server made by Matthew on 13 February. Apart from conveying status updates, comments are sometimes used more directly for coordination, as in "be sure to update before doing any changes" (comment made by Justin on 5 May). The timeline is thus central to the coordination of SD work in the project, particularly when team members are not collocated. It provides an instant overview of state-of-affairs as well as an entry point to the project artifacts.

Research Method

The research reported in this paper is interpretive [48]. The main source of data is a retrospective reflection workshop conducted with the selected SD project team in a usability lab over a period of two and a half days at the end of their project. In addition, a longitudinal study of the team's work had been conducted throughout the project (one semester), including 45 h of non-participant observation of meetings

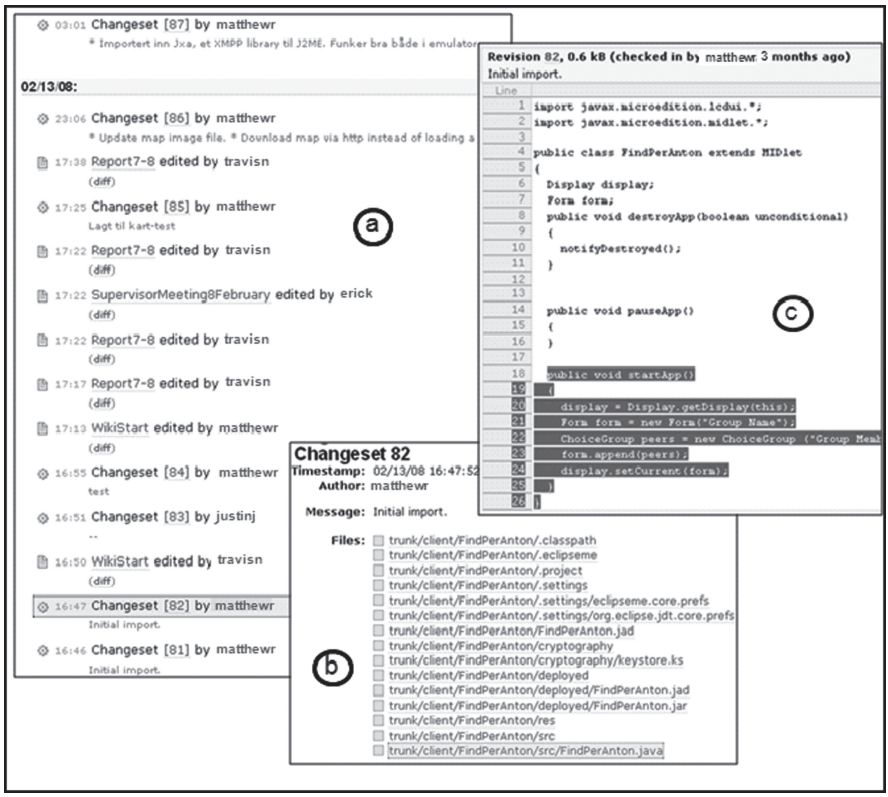

Fig. 2 Three screenshots from Trac as Justin examines the timeline and identifies pre-study coding

and work in the computer lab and frequent examination of project artifacts (including the contents of Trac) as well as the team's internal and external email communication. The study provided knowledge important to our interpretation of data from the workshop. Selection of the project was based on a combination of the technically complex project task and the team's decision to use Trac.

The workshop steps were based on the timeline and satisfaction curve approach described in the Case section. Compared to the short retrospective workshops conducted with the other teams in the course, the design incorporated an extra step of using historical data in Trac to aid recall. Further, individual timeline construction was done in a sequence of individual sessions (rather than in parallel) to allow the collection of rich data about each team member's timeline construction and explore how the use of historical data in Trac might be helpful in different ways or to different degrees among the team members. The lab was equipped like a small meeting room with a whiteboard and a PC. Several movable cameras were installed in the ceiling, and all sessions were videotaped. The use of Trac was recorded with screen capture and synchronized with the video recording. Photos were taken of the whiteboard.

In line with the principle of considering multiple interpretations [38], individual follow-up interviews were made 5 months after the workshop, i.e. when analysis of the data had resulted in findings to be presented to the participants and discussed in light of their interpretation.

Two questions guided our focus in the workshop and the subsequent data analysis. First, we looked for indications that the retrospective use of Trac benefited reflection, e.g. helped the team return to experience, re-evaluate it and create new perspectives. More specifically, we looked for events recalled *only by some team members*, events recalled *only after use of Trac* and, among those, events *of general importance in later discussion*, e.g. addressing lessons learned. Second, we wanted to know more about *how* Trac was used by the team members to aid their recall and reflection, i.e. what particular functionality and data in the tool seemed relevant and useful. In the data analysis we systematically examined the individual timelines and the shared timeline and the recordings of the sessions in which they were made, making a table of all events that were mentioned, noting who had mentioned them and whether they were mentioned before or after examination of the data in Trac. We picked *one* event, namely pre-study coding, based on the following criteria: the event was recalled only after use of Trac, it was recalled by some team members only, and it was important in the team's final discussion of lessons learned, The selection of an event guided further analysis, including close examination of all workshop data relating to the event with the aim of to providing a coherent account of how the event was elaborated in, and impacting on, the reflection process in the team, Data were transcribed and translated into English at need. In the presentation, names (including user names in screenshots) are made anonymous.

Our study can be seen as a "fine-grained, field-based empirical study" useful for designing systems and understanding the nuances of practice [24]. Even though the reflection workshop in our case was organized with a clear research agenda, impacting e.g. on its duration, the participants did real reflection on their SD project.

There are two main limitations to the study. First, it was conducted in the context of SD education. The project was however close to industry practice, e.g. with a customer and requirements for a working software product, and the team had some members who were relatively experienced as developers. Second, a single case study limits the possibilities to generalize from results. However, our purpose was to understand the details of a team's reflection on their project and investigate the potential to support reflection through certain steps and with certain resources.

Findings

In this section we draw on our workshop data and describe the project team's recall of and reflection on 'pre-study coding', an example of a project event recalled only through the aid of historical data in the collaboration tool. To provide some context, we start by describing the team's project process on basis of the longitudinal study.

The Process and Organization of the Case Project

The members were generally satisfied with their team and the assigned task. The relationship with supervisor and customer worked well throughout the project. Requirements for the product were more or less clear after the second customer meeting. Prior to midterm report delivery, time in the project was spent partially on trying out technology, later to be characterized by the team as 'pre-study coding', partially on writing the project report. After four iterations, the team delivered a product with which both they and their customer were satisfied.

The team had decided on a flat organization, formally with Eric as project manager. In practice, Justin had that role, taking main responsibility e.g. for deliveries and meeting minutes. Matthew was seen as the technical expert, taking responsibility for the server application and the overall design. Justin was in charge of the client application, and Travis for the map functionality implemented late in the project. All five team members actively participated in programming and report writing. Figure 3 (created by the authors processing information from the Trac timeline) shows that all team members participated and that Matthew and Justin did the most ticket updates.

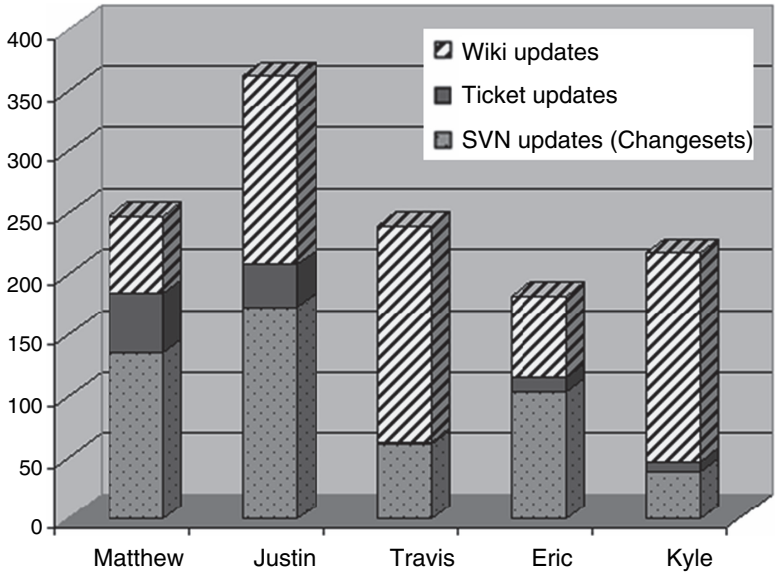

Fig. 3 A summary overview of the Trac timeline contents

Pre-study Coding in the Individual, Memory-Based Accounts

The five individual whiteboard timelines created on the basis of memory alone (workshop Step 1, see Table 1) vary in terms of the selection of events and the shape of the satisfaction curves. The events remembered to some extent reflect the roles in the project. For instance, Matthew is the only one who remembers choices of certain server technology. In comparison, a cancelled feature freeze that had a high impact on the entire team, is remembered by four out of five members. The onset of pre-study coding is not mentioned as important by anyone at this point. The onset of 'coding' is however mentioned by Matthew and Justin. Matthew refers to a couple of events involving the choice of technology in the project before midterm, but says nothing explicitly about coding in this period. He places 'coding startup' right after midterm. Justin places 'code start' on his timeline just before midterm. "Before that we had only been working with report, and then we actually started writing code." When accounting for the impact of a necessary change of framework some time after midterm, Justin explains that most of the coding of the final product took place during a short period towards the end of the project.

Pre-study Coding Emerging Through Examination of Trac

With the help of Trac in Step 2, all team members identify new events. In addition, there are some modifications of names of existing ones and changes of their timeline position. Justin and Matthew are the team members for which the examination of the timeline brings up most new events. As they chronologically examine the timeline, Justin and Matthew both examine the changesets [81] and [82] made by Matthew on 13 February. The comment 'Initial import' associated with both changesets indicates that they are about the inclusion of files needed to start producing and testing Java code. Figure 2 shows the sequence of screenshots as Justin investigates the timeline item of changeset [82]. Starting from the timeline view (a), he clicks [82]. The window (b) now displays what has been changed; a list of files. As a programmer Justin knows that the coding done by Matthew is in FindPerAnton.java, the rest of the files being files necessary to set up a running program and included for the first time in the initial import. He clicks the filename, and the window changes to show the file contents (c). Examining the code, Justin exclaims: "Oh yes, it was only, kind of 'Hello world' on the mobile" and returns to the timeline (a). 'Pre-study coding' is then added to the whiteboard timeline (see Fig. 4).

Matthew, on examining the same item during his timeline construction, says: "Right there, we have started coding. It was probably pre-study coding." He suggests that the event be placed in the first half of the pre-midterm project period (see Fig. 4). As a consequence, 'startup coding', already on Matthew's timeline just after midterm, is changed to 'startup coding according to requirements spec (start iterations)'.

Table 1 Organization of the reflection workshop

Session type	Step	Explanation
Individual sessions (Days 1–2, each session 1.5–2 h) *The whiteboard was photo-graphed after these sessions*	(1) Reconstructing the project trajectory by memory	The team member was given a sheet of paper with a timeline containing a few fixed project events (e.g. delivery deadlines) and was asked to add important events. Next, for each of these events, the researcher added it to a similar timeline on the whiteboard and asked the team member to explain the importance of the event. Next, the team member was asked to draw on the paper sheet a curve showing how he had felt about working in the project. Finally he was asked to draw the curve along the whiteboard timeline, explaining its shape.
	(2) Using collaboration tool history to aid further reconstruction	The team member was asked to use the laptop and Trac to freely explain the project. When general browsing and explanation appeared exhaustive, the team member was asked to do a chronological walkthrough of the Trac timeline, exploring links at need. As new events deemed important to the project were identified, they were added to the whiteboard timeline by the interviewer, who took care to add events based on what *the team member* considered important.
Common session (Day 3, half day)	(3) Reconstructing the project trajectory using collaboration tool at need	The project timeline was reconstructed again, the researcher writing down events on the whiteboard as the team members listed them in a round-robin fashion. A PC with access to Trac was available to the team members, the screen projected onto the wall next to the whiteboard. When no more events were suggested, the team members came to the whiteboard one by one to draw their satisfaction curve and comment on it.
	(4) Discussion of tasks, lessons learned, roles	There was a common session addressing questions of project roles, tasks and lessons learned, each question answered by all team members in turn. Discussion was allowed and encouraged.

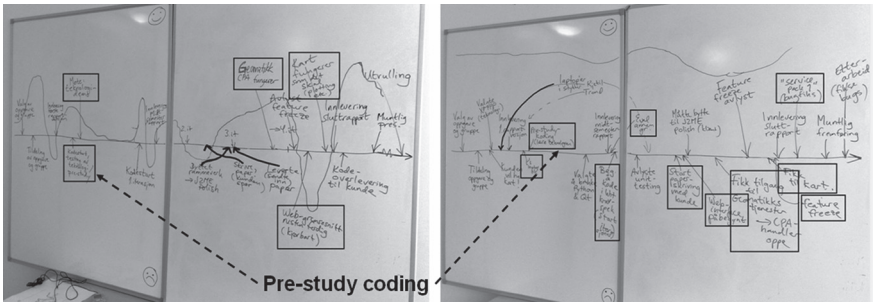

Fig. 4 Justin's and Matthew's timelines. Frames/emphasis of *arrows* show additions/modifications after examination of Trac

Pre-study Coding in the Co-constructed Account

In the collective session of Step 3, most events from the individual timelines were included in the shared timeline. Occasionally, the team on their own initiative looked into Trac to check details, typically the timing of events. The issue of pre-study coding was explicitly addressed. Justin early on repeated his statement that coding did not start until midterm, and this was not questioned by anyone. This can be interpreted as a shared understanding in the team that 'coding' (as a standalone term) meant 'coding in accordance with requirements specification'. As part of the round-robin provision of events, 'pre-study coding' was mentioned by Justin and added to the timeline. Eric suggested another name for the same event: "Unofficial coding". This received no further comments. In the later step of drawing satisfaction curves along the whiteboard, Justin explicitly referred to pre-study coding as he made an upward curve bend early in the project, explaining that he enjoyed the pre-study coding because they got things working then.

Pre-study Coding in Reflections on Lessons Learned

The topic of pre-study coding was recurrent in the discussion in Step 4, particularly when the team addressed lessons learned. Justin suggested that they might have overrated their own capabilities by starting coding as late as midterm. Matthew said that more pre-study coding might have helped the team avoid the extra work of learning a new framework, because they would have known better the limitations of the technology. Overall in the discussion, pre-study coding was given a central role in the account of what happened in the project and in conceptions of how to ideally run a similar project. The considerations on pre-study coding were made by Justin and Matthew. There seemed to be full agreement in the team that

pre-study coding is necessary to reach the technical competence needed for producing design and code when a development task requires use of technology new to the team.

Insights from the Follow-Up Interviews

In the follow-up interviews with the team members, they were presented with the researcher's interpretation that the workshop had lead to a shift in the team's attention to pre-study coding and conception of its significance to the project. All team members expressed that this finding is in line with their own interpretation of what happened in the workshop. The interviews with Eric, Travis and Kyle however revealed some viewpoints that had not been brought into the team's discussion. When Eric during the collective reconstruction of the timeline had used the term 'unofficial coding' about the pre-study coding, he did not refer to the fact that the team de-emphasized this activity but to the fact that pre-study coding had been conducted by some team members only. In Eric's opinion, the project would have gained from a more distributed effort, making the entire team more competent with the technology at an earlier point. Travis and Kyle expressed similar views. While these viewpoints might have matured in the period after the reflection workshop, the workshop recordings indicate that the views were there during the workshop but had never properly entered the discussion.

Discussion and Implications

The example in the Findings illustrates how collaboration technology supporting day-to-day work practice in a SD project contained data that could be utilized by team members in a systematic and collaborative effort to reconstruct and reflect upon the project process.

The event called 'pre-study coding' was *identified only through the aid of collaboration tool history* and *only by two of the five team members* during the individual timeline construction. The event was later brought into the team's shared timeline, was referred to by a team member as impacting on his personal experience of the project, and was central in the team's reflections on lessons learned. Comparing the team members' original accounts of the project at the outset of the workshop with the final discussion, there was a clear difference: the significance of pre-study coding, in this project in particular and in SD projects in general, had become clearer. We see this as an example of knowledge relevant to the SD practice being created through retrospective reflection.

In this section we address why the tool-aided retrospective reflection was successful. We do this by considering the work practice, including the retrospective reflection effort itself, as a case of distributed cognition [18], and by

looking into the type of historical data available in the tool, the tool features used to retrieve the data in the reflection workshop, and the organization of the workshop.

Historical Data in the Collaboration Tool

Cognition involved in a work practice is distributed over participants and the artifacts involved. Accordingly, it will typically be necessary to examine different data sources to make sense of an issue or event associated with that practice. In our case, what made Justin recognize that coding had happened in the project, was an item in the Trac timeline (see Fig. 2). The *type* of timeline item (a change to a file in the underlying file versioning system) and the *comment* following the change (i.e. the comment explicitly added by the user making the change, for purposes of day-to-day coordination) made it possible for him to quickly infer what this was about. Following the link, he recognized the element of interest in the list of files, and selecting the file in its there-and-then version, he could see what the piece of code was actually doing. The combination of the data and Justin's background (as a programmer, Trac user, and co-creator of the data in front of him), enabled him to reconstruct the state of the project and identify the event.

The type of data involved included (1) data used for day-to-day *coordination of project work*, reflecting ongoing tasks (e.g. timeline items with comments, made by different team members; the list of artifacts to which the selected timeline update applies) (2) data describing the *state of an artifact in the project workspace* (e.g. the there-and-then version of the source code file). Different aspects of the development work (e.g. project management, coding) being distributed in accordance with team roles, the data used to identify pre-study coding reflects a *social distribution* of the cognitive processes involved in the onset of pre-study coding.

The combination of data originating in coordination and data from the development workspace is useful for identifying events associated with development. A type of data less relevant for the identification of pre-study coding, but that frequently evoked team members' recall and reaction in the workshop of our study, are timeline comments addressing the social/emotional side of work and thus providing *social awareness* in day-to-day work.

Comparing the data examined in the workshop sequence illustrated in Fig. 2 with the aggregate data on the project process shown in Fig. 3, we note that the aggregate data could not have been used to provide context for single events. The diagram, though a useful resource, provides only a partial overview of the process. In our case, the aggregate diagram shows that all team members participated. What *cannot* be seen from the diagram is the exact type and amount of work done by each team member. For instance, we knew from observation of the team that Matthew tended to make updates in large increments, uploading new code to the shared server after working on it locally on his own machine for long, sometimes for days. While the diagram thus seems to indicate that Matthew and Eric did a similar

amount of coding in the project, Matthew in reality did more coding than Eric. Another example is that when two team members were working in pairs, one of them tended to be in charge of the keyboard.

In sum, the aggregate timeline data provides an additional overview. To get adequately detailed and contextualized historical data from Trac for the recall of specific project events, however, an examination of individual timeline items is necessary. It should also be noticed that the fact that project work was sometimes conducted in pairs is not reflected *anywhere* in Trac. The day-to-day usage of the tool provides it with potential to be used in shedding light on some aspects of the project process – but not all.

Navigating the Historical Data in the Collaboration Tool

It was through chronological *traversal* of the timeline events that Justin identified the first uploading of source code to the shared workspace. Also, the timeline provided a condensed *overview*, including the possibility – unexploited in this case – to filter the type of events displayed. From the timeline item of interest, it took just a click to get more detailed, but still overview level information about the change, e.g. which files had been changed. The usefulness of this feature fits with the point made in [45] that missing links from bug records to source code is highly problematic for the reconstruction of the bug fixing process. Going *up and down between different levels of detail* thus proved important. From the overview, there was direct access to the *state* of the artifact (e.g. the selected file) at the time of the change. *Switching* between the modes of traversal, overview, and exploring the state of the artifact was easily achieved. The ease with which several months' work can thus be navigated in Trac by participants contrasts with the effort needed to examine less structured data on collaborative work, e.g. video recorded meetings [49].

While the data in the tool reflects socially distributed cognition, the timeline reflects the temporal distribution of cognition and is a starting point for identifying significant events. Another aspect of the temporarily distributed cognition is that when the tool is used retrospectively to make sense of data, familiarity with the tool provides context for the user. Justin had been using the Trac timeline daily throughout the project to get an overview of current project status. Justin's experience and skills in *how to make sense of the process through the collaboration tool* mediated his retrospective reflection.

Appropriately Organizing the Process of Retrospective Reflection

Essential to the success of the reflection workshop was its *organization*. The example in Fig. 2 shows *individual* use of Trac to reconstruct a timeline, but as a whole, our study shows how historical data successfully plays a role in supporting

cooperative work. A core idea in the approach [8, 9] on which the workshop was based is that individual perspectives should be utilized in the construction of a shared understanding. In the study, individual timelines mediated individual recall and reflection as well as collective timeline construction.

From a distributed cognition perspective, the individual and shared timelines can be seen as external representations of the project trajectory, and the steps of the workshop as a process of coordinating the cognitive processing between internal and external representational states. In our study, we looked at a collaboration tool as a resource for the cognitive processing between internal and external representational states. This happens in three ways, as illustrated in Fig. 5. The day-to-day work, seen as a process trajectory along which there are multiple points of action and interaction, leaves a trace of external representations of the distributed cognition involved, in the form of historical data in the tool. This is a result of Trac's role in *supporting the day-to-day cooperative project work* (1). In the process of retrospective reflection, Trac is used as an aid to *individuals' recall of project events* (2), resulting in the creation of the individual timelines, which are also external representations of the project trajectory. The individual timelines are next used as a resource when the team collaboratively creates another external representation of the project trajectory: the shared timeline. To aid the alignment of the individual timelines, Trac is used collaboratively by the team as an aid to *clarify details* in the construction of a shared representation of the project (3), e.g. exact dates and times

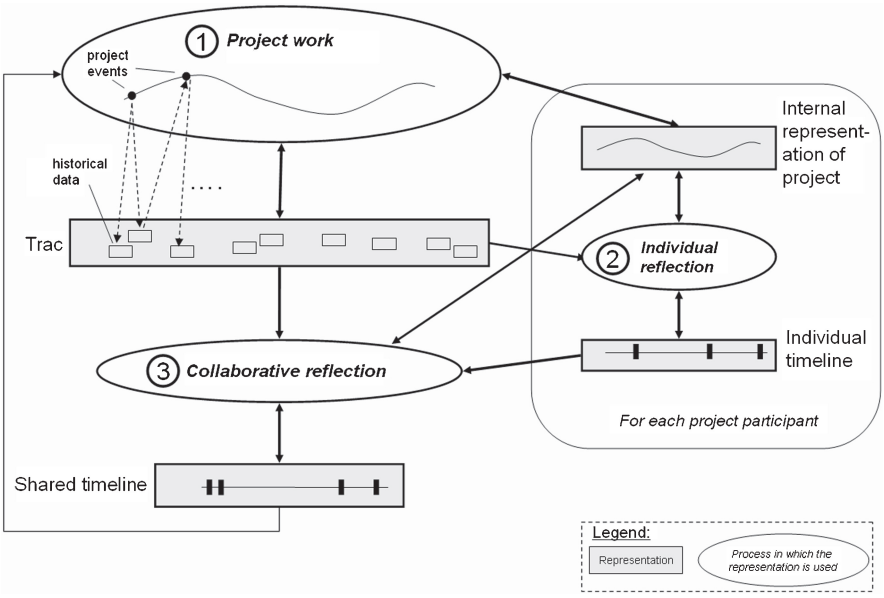

Fig. 5 Trac in a dual role as a collaboration tool in a project: supporting day-to-day work *and* retrospective reflection on that work (Adapted from [18])

of specific events. Figure 5 is an instantiation of the more generic model of retrospective reflection outlined in [18].

Used in retrospective reflection on day-to-day development work the collaboration tool serves as a boundary object between that practice and retrospective reflection [50]. Developers' awareness that the tool will be used retrospectively may affect their day-to-day work practice [24]. For instance, software developers may start providing more informative comments about their updates to make it easier to understand them retrospectively. This is likely to benefit day-to-day coordination. A change with potentially negative effects is if developers, to make a good retrospective impression of their work, start hiding information. For instance, mistakes in intermediate stages of work can be hidden by making less frequent updates or by avoiding commenting about error corrections (a type of comment common in the Trac timeline in our study). This could adversely affect day-to-day workspace awareness as well as the possibility to get a realistic picture of the development process from the Trac timeline.

Our findings indicated that the team in our study through the tool-aided retrospective reflection succeeded in creating new knowledge about their software development practice. However, the individual differences in respect of how helpful Trac was in the recall of events, point to challenges of uneven power distribution in the team. The benefit of using a development-oriented collaboration tool to aid reflection appeared greater for those dominating the development work. Strong involvement in the activity from which the data originated provided them with context for recognizing the data as connected to project events. Having *identified* the pre-study coding event, the two lead programmers seemed to retain ownership of the event in the team's discussion, which can explain why alternative viewpoints on pre-study coding among some team members were never properly disclosed.

The Potential of Collaboration Tools to Support Reflection in SD Work: A Set of Guiding Questions

The previous sections outlined how certain tool features as well as the type of data stored in the tools and the organization of the reflection activity were important when historical data in Trac was used to aid the reconstruction of the project trajectory in retrospective reflection. In this chapter we ask: in what way are these findings useful from a perspective of supporting reflection on SD work more generally?

Our answer is twofold. First, we believe that the results hold promise for the use of *tools like Trac* to assume a dual role of supporting daily work as well as retrospective reflection in SD projects. It was relatively unproblematic in our study to include steps of investigating collaboration tool history into an existing approach to organizing retrospective reflection. To make the tool-aided version of the approach integral to SD practice, the workshop organization must be adjusted to feasible schedule, i.e. one that can be followed under resource constraints in real projects.

Second, we believe that the findings can be generalized to the use of collaboration tools in retrospective reflection in SD work, answering the intent to provide empirically grounded insights to project teams and tool designers about what makes collaboration tools useful for such a purpose. In Section Discussion and Implications, we structured the findings into some issues that seem crucial to the success of the selected collaborative tool to aid retrospective reflection. Based on this understanding, we outline questions that point to essential aspects of how a collaboration tool and its historical data can support retrospective reflection. The objective is not to find the 'best' tool for the support of reflection but to help users and designers identify, or modify, strengths and weaknesses of the tools for such use. To illustrate how the answers may differ for various lightweight collaborative tools, we refer to the research literature on some tools often used in project work: project wikis, instant messaging, and email.

What aspects of the project process are reflected in the historical data in the tool? Which SD challenges worth reflecting upon have been addressed by use of this tool? An issue tracker contains data that may be used to aid reconstruction of the development process, with a focus on its technical and formal aspects. The *informal* communication is likely to be reflected in other tools, e.g. instant messaging tools [28, 51]. Other lightweight project management tools, e.g. project wikis [52], might reflect more of the informal collaboration in the team than what is documented in the historical data of an issue tracker. Stakeholder collaboration, a frequently challenging SD issue [36], may partially be reflected in email logs.

Does the tool provide features for getting a chronological overview of aspects of the project process? As seen in our study, the traversal of a chronological overview can be used to structure the process of examining historical data. Chronological overviews are essential in project management and thus provided by project management tools such as Trac. A project wiki, by being a wiki, contains functionality for getting an overview of the revisions of each page [52]. The disadvantage of browsing through numerous revisions of a wiki page is that changes may be small and uninteresting. Conversely, increments reflecting the process *as it unfolded* may shed light on the process in a way complementing more formal overviews. An email application allows for the filtering of email messages to be displayed in a mailbox, enabling for instance rapid overview of all email sent to the project customer or all messages with a particular keyword in its title (e.g. 'status report'). IM tools generally provide poor overview information, but the log contents are chronological and time-stamped.

Does the tool provide features for accessing the project artifacts in their there-and-then versions? One of the strengths of Trac as a SD tool is the way task organization links to project artifacts, stressing the close connection between process and product in SD work, the complexity of the process depending on the complexity of the product [1]. A project wiki may provide direct access to some project artifacts [17], but is not likely to provide read access to the entire development-related workspace with its artifacts in their there-and-then versions, as does Trac. Email frequently includes relevant project artifacts as attachments. Instant messaging logs might contain elements of project artifacts (e.g. source code excerpts) [51].

Does the tool provide features for easy navigation between overview and detail? The possibility to go into detail on a specific project event helped the participants in our study recognize it as an event worth including in their project timeline. Looking at project wikis, they provide reasonably good support for this e.g. through the page history overviews with links to each specific page [52]. Navigation in an email reader can be more cumbersome, but typically allows e.g. the use of different windows for the mailboxes and the contents of individual messages. Instant messaging logs mainly allow navigation on a detailed level of project interaction only.

With respect to overviews and navigation, limited features in the collaboration tools used to create the data may be amended by the use of other lightweight tools (e.g. search tools) to navigate the data. However, this comes at the loss of the contextualization of the data in the work (tool) environment from which it originated, a loss which may negatively affect participants' sense-making of the data.

Are the historical data subject to privacy issues? While the study reported in this paper considered data that were regarded as shared within the team, these characteristics do not apply to all data from project collaboration. IM logs, for instance, may be considered by their owners as too private to be shared [18]. Their potential to support *individual* reflection on the project process may be considered.

How will the use of historical data from the tool be integrated into a structured reflection activity with individual and/or collaborative sense making enabling participants to return to experience, reconstruct the stories [45] and draw lessons learned from the data? The organization of the activity can for instance be based on SD best practice for project retrospectives [8, 9]. An issue which is not the main focus in this paper, but which is nevertheless essential and should be considered as part of the organization of retrospective reflection, is how to subsequently draw upon the lessons learned in the work practice, within and/or across projects and teams.

For a project team considering the potential of one or more of their collaboration tools to support their reflection on, and learning from, their project process, it may be a good start to consider what aspects and challenges of their project work they would like to shed light on, and next consider what are the candidate tools to contain relevant data.

Finally, it should be stressed that when considering the potential of using a certain tool to aid retrospective reflection *in a specific case*, a general framework is only a useful starting point. The functionality of the specific tool will have to be considered along with the usage of the tool in the specific project, which heavily impacts on what historical data is being stored through everyday use of the tool.

Conclusion

By investigating a case of retrospective reflection in a software development project aided by historical data in an issue tracker (Trac), we have shown in detail how a collaboration tool in daily use in a project can be used to help participants learn

from the project experience. We have demonstrated how certain types of data and certain features for retrieving them proved particularly valuable to aid participants' recall and reflection, aiding the reconstruction of the project trajectory.

Based on the findings we identified a set of questions that may be asked by a project team or a collaboration tool designer when considering the potential to use specific collaboration tools in retrospective reflection. The questions can serve as a starting point for the development of a more general framework outlining issues that should be addressed. To gain insight on the potential of different types of collaboration tools to aid retrospective reflection, more empirical studies are however needed. By unveiling project teams' use of specific tools in retrospective reflection we may identify ways of utilizing the tools that significantly differ from the way Trac was used in our study. This is an area of further research.

Another issue to be addressed in further research is how a project team's knowledge of the future use of a collaboration tool in retrospective reflection affects their daily use of the tool.

Finally, we will examine how the retrospective use of Trac can be integrated into retrospective reflection workshops within a time schedule feasible for real SD projects. We will do this by trying out the approach in SD projects on a larger scale.

References

1. Carstensen, P.H. and K. Schmidt, Computer Supported Cooperative Work: New Challenges to Systems Design, in Handbook of Human Factors/Ergonomics, K. Itoh, Editor. 2002 (1999), Asakura: Tokyo.
2. Grinter, R. Doing Software Development: Occasions for Automation and Formalisation. in ECSCW'97. 1997. Lancaster, UK: Springer.
3. Herbsleb, J.D., et al. Distance, dependencies, and delay in a global collaboration. in CSCW'00. 2000. Philadelphia, PA: ACM.
4. Keil, M., J. Mann, and A. Rai, Why Software Projects Escalate: An Empirical Analysis and Test of Four Theoretical Models. MIS Quarterly, 2000. 24(4), pp. 631–664.
5. Lyytinen, K. and D. Robey, Learning failure in information systems development. Information Systems Journal, 1999. 9: p. 17.
6. Basili, V.R. and G. Caldiera, Improving Software Quality by Reusing Knowledge and Experience. Sloan Management Review, 1995, pp. 55–64. Fall 1995.
7. Dybå, T. and T. Dingsøyr, Empirical studies of agile software development: A systematic review. Information and Software Technology, 2008. 2008(50): pp. 833–859.
8. Derby, E., D. Larsen, and K. Schwaber, Agile Retrospectives. Making Good Teams Great. 2006: Pragmatic Bookshelf. Raleigh, North Carolina.
9. Kerth, N., Project Retrospectives: A Handbook for Team Reviews 2001: Dorset House, New York.
10. Schindler, M. and M.J. Eppler, Harvesting project knowledge: a review of project learning methods and success factors. International Journal of Project Management, 2003. 21: p. 10.
11. Bjørnson, F.O., A.I. Wang, and E. Arisholm, Improving the effectiveness of root cause analysis in post mortem analysis: A controlled experiment. Information and Software Technology, 2009. 51: pp. 150–161.
12. Schön, D., The Reflective Practitioner. 1983: Basic Books, New York.
13. Kasi, V., et al., The post mortem paradox: a Delphi study of IT specialist perceptions. European Journal of Information Systems, 2008. 17: pp. 62–78.
14. Churchill, E.F. and S. Bly. It's all in the words: Supporting work activities with lightweight tools. in GROUP '99. 1999. Phoenix, AZ: ACM.

15. Gutwin, C., R. Penner, and K. Schneider. Knowledge sharing in software engineering: Group awareness in distributed software development in CSCW'04. 2004. Chicago, IL: ACM Press.
16. Handel, M. and J.D. Herbsleb. What is Chat Doing in the Workplace? in CSCW'02. 2002. New Orleans, LA: ACM.
17. Krogstie, B.R. The wiki as an integrative tool in project work. in COOP. 2008. Carry-le-Rouet, Provence, France: Institut d'Etudes Politiques d'Aix-en-Provence.
18. Krogstie, B.R. A model of retrospective reflection in project based learning utilizing historical data in collaborative tools. in EC-TEL 2009. 2009. Nice, France: Springer.
19. Hutchins, E., Cognition in the Wild. 1995, Cambridge, MA: MIT Press.
20. Rogers, Y. and J. Ellis, Distributed Cognition: an alternative framework for analyzing and explaining collaborative working. Journal of Information Technology, 1994. 9: pp. 119–128.
21. Daradoumis, T. and M. Marques, Distributed Cognition in the Context of Virtual Collaborative Learning. Journal of Interactive Learning Research, 2002. 13(1/2): pp. 135–148.
22. Flor, N.V. and E.L. Hutchins. Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming during Perfective Software Maintenance. in Empirical Studies of Programmers: Fourth Workshop. 1991: Ablex Publishing, Norwood, NJ.
23. Sharp, H. and H. Robinson, Collaboration and co-ordination in mature eXtreme programming teams. International Journal of Human-Computer Studies, 2008. 66(7): pp. 506–518.
24. Ackerman, M.S. and C. Halverson, Organizational Memory as Objects, Process, and Trajectories: An Examination of Organizational Memory in Use. Computer Supported Cooperative Work, 2004. 13(2): pp. 155–189.
25. Salomon, G., No distribution without individuals' cognition, in Distributed Cognitions. Psychological and educational considerations, G. Salomon, Editor. 1993, Cambridge University Press, Cambridge.
26. Strauss, A., Continual permutations of action. 1993, New York: Aldine de Gruyter.
27. Grudin, J. and T. Lovejoy. Messaging and Formality: Will IM Follow in the Footsteps of Email. in INTERACT 2003. 2003. Zurich: IOS Press.
28. Isaacs, E., et al. The Character, Functions, and Styles of Instant Messaging in the Workplace. in CSCW'02. 2002. New Orleans, LA: ACM.
29. Dourish, P. and V. Bellotti. Awareness and coordination in shared workspaces. in ACM CSCW. 1992. Toronto, Ontario, Canada: ACM Press.
30. Gutwin, C. and S. Greenberg, A Descriptive Framework of Workspace Awareness for Real-Time Groupware. Computer Supported Cooperative Work 2002. 11(3–4): pp. 411–446.
31. Gutwin, C., R. Penner, and K. Schneider. Group Awareness in Distributed Software Development. in CSCW'04. 2004. Chicago, IL: ACM.
32. Bødker, S. and E. Christiansen, Computer Support for Social Awareness in Flexible Work. Computer Supported Cooperative Work, 2006. 15: pp. 1–28.
33. Schmidt, K. and C. Simone, Coordination Mechanisms: Towards a Conceptual Foundation of CSCW Systems Design. Computer Supported Cooperative Work, 1996. 5: pp. 155–200.
34. Johnson, J.N. and P.F. Dubois, Issue Tracking. Computing in Science and Engineering, 2003(November/December): pp. 71–77.
35. Prause, C.R., et al. Managing the Iterative Requirements Process in a Multi-National Project using an Issue Tracker. in IEEE International Conference on Global Software Engineering. 2008. Bangalore, India: IEEE.
36. Poole, W.G. The Softer Side of Custom Software Development: Working with the Other Players. in Conference on Software Engineering Education and Training. 2003. Madrid, Spain: IEEE.
37. Kraut, R.E. and L.A. Streeter, Coordination in Software Development. Communications of the ACM, 1995. 38(3), pp. 69–81.
38. Whittaker, S., D. Frohlich, and D.-J. Owen. Informal Workplace Communication: What Is It Like and How Might We Support It? in Human Factors in Computing Systems. 1994. Boston, MA: ACM Press.
39. Dekel, U. and J.D. Herbsleb. Pushing Relevant Artifact Annotationsin Collaborative Software Development. in CSCW'08. 2008. San Diego, CA: ACM.
40. Storey, M.-A., et al. Shared waypoints and social tagging to support collaboration in software development. in CSCW'06. 2006. Banff, Alberta, Canada: ACM.

41. Fitzpatrick, G., P. Marshall, and A. Phillips. CVS integration with notification and chat: lightweight software team collaboration. in CSCW'06. 2006. Banff, Alberta, Canada: ACM.
42. Cubranic, D., et al. Learning from project history: a case study for software development. in CSCW '04. 2004. Chicago, IL: ACM.
43. Burge, J. and D.C. Brown. SEURAT: integrated rationale management. in ICSE'08. 2008. Leipzig, Germany: ACM.
44. Burge, J. and J. Kiper. Capturing Collaborative Design Decisions and Rationale. in Design, Computing, and Cognition. 2008. Atlanta, GA, USA.
45. Aranda, J. and G. Venolia. The Secret Life Of Bugs: Going Past the Errors and Omssions in Software Repositories in International Conference on Software Engineering (ICSE'09). 2009. Vancouver, Canada: IEEE.
46. Trac home page, http://trac.edgewall.org/, Last accessed 10 September 2009.
47. Krogstie, B.R. and M. Divitini. Shared timeline and individual experience: Supporting retrospective reflection in student software engineering teams. in CSEE&T 2009. 2009. Hyderabad, India: IEEE Computer Society.
48. Klein, H.K. and M.M. Myers, A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. MIS Quarterly, 1999. 23(1): pp. 67–94.
49. Wolf, C.G., J.R. Rhyne, and L.K. Briggs. Communication and Information Retrieval with a Pen-based Meeting Support Tool. In: CSCW 1992, Toronto, Canada, ACM.
50. Star, S.L., The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving, in Distributed Artificial Intelligence, M. Huhns and L. Gasser, Editors. 1990, Morgan Kaufmann, Menlo Park, CA. pp. 37–54.
51. Krogstie, B.R. Do's and dont's of instant messaging in students' project work. in NOKOBIT 2009. 2009. Trondheim, Norway: Tapir.
52. Krogstie, B.R. Using Project Wiki History to Reflect on the Project Process in 42nd Hawaii International Conference on System Sciences. 2009. Big Island, HI: IEEE Computer Society.