An Educational Testbed for the Computational Analysis of Collaboration in Early Stages of Software Development Processes

Thomas Kowark, Jürgen Müller, Stephan Müller, Alexander Zeier
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract

Agile software development processes are widely adopted in software engineering projects. Their low organizational overhead and iterative nature make them ideal choices for small development teams. The application of those methods in software projects that require collaboration between multiple sub-teams is a challenging task that remains subject to intensive research. Especially the initial phases of such projects are crucial for project success since a problem-free inception period generates a basis for efficient development later in the process.

We introduce a testbed that allows analyzing collaboration processes during those early stages of software development within a low-risk, educational setup. Participants of a software engineering lecture form development teams of considerable size and develop real-life applications in a realistic, yet controlled, environment. By combining manual observations with the computational analysis of digital collaboration artifacts we are able to gain insights into distinctive patterns of collaboration activity and reason about their triggers within the process setup.

1. Introduction

During the last decade, agile methodologies and development processes became the most natural and widely adopted solution for software development projects in small teams [2]. A wide variety of such processes exists, each tackling another aspect of the problem domain. Accordingly, hybrid forms of different process models have been developed to combine different benefits to new, more efficient processes.

In contrast to the usage of agile methodologies in small projects, large-scale adoption still remains subject to intensive research [3], [16], [22], [23], [25], [33]. It is well-defined how activities such as knowledge sharing or coordination of different development efforts should be performed within groups of up to ten people, but new challenges arise when multiple sub-teams develop one target system in a joint effort. Means have to be found to ensure that all teams share a common vision of the project, communicate gained knowledge amongst each other, and develop parts of the system that, in the end, work together seamlessly. Shared workspaces, conferencing, or coaching are just some examples that have proven to be beneficial for agile software development projects involving multiple interlinked teams [24], [34], [40]. Especially during early project phases, it is crucial to foster close collaboration between the teams, since differing comprehension and implementation of user needs have detrimental effects that pervade and amplify over the lifetime of the project [4].

Companies repeatedly share insights gained within large, agile development projects (e.g., [5], [19]), but testing the conclusions drawn from analyzing those case studies is rarely possible since the project setup and team composition are unique to that particular case and will usually not be repeated by the same company to compare different process models or collaboration techniques. In academia on the other hand, it is not only possible but common to perform similar and comparable projects on a regular basis as part of the curriculum.

Therefore, we designed the exercise of a university lecture to resemble the initial stages of a software development process that involves a substantial amount of developers working in small but highly interlinked development teams to develop a single system in a joint effort. The project includes a variety of common
software engineering activities, such as requirements engineering, user research, or testing, and uses widely adopted open-source groupware tools for inner-project collaboration.

As previously mentioned, analysis of collaboration behavior is the main focus of our research. In our context, collaboration comprises all activities that contribute to sharing project-relevant knowledge between project members. These activities are, especially in agile projects, predominantly verbal and non-verbal face-to-face interactions, but to a certain degree also manifest themselves in digital artifacts, such as emails, wiki pages, bug tracker items or source code revisions.

Accordingly, while we are analyzing the digital collaboration artifacts that the student teams generate throughout the projects, we also strive to gain deeper insights into the team dynamics by using a combination of computational analysis and manual observation through senior students with prior project experience. This way, it is possible to identify potential indicators for bottlenecks in the inter- and intra-team collaboration and observe effects of chosen countermeasures.

In addition to help gaining insights into aspects of collaboration in software development processes, the course also offers a unique learning experience for the students. Projects of comparable size are rather uncommon in academia and can mostly be found in large companies. Thus, the exercise additionally contributes to a better preparation of students for professional software development [7], [17], [20], [32], [41].

The testbed that we present in this paper has been evaluated in an initial case study. Based on the gained insights we are continuously improving and extending the observation tools and identify aspects of the process that have to be closer investigated in future iterations of the lecture. Ideally, the general idea of the testbed and our toolset can be exported to other universities and, in turn, increase the database for our studies.

The remainder of this paper starts with a detailed description of the testbed characteristics and our observation techniques. After a presentation of the setup and the results of the first conducted case study, we give an overview about related work and present the next steps of our research.

2. Testbed

The following chapter presents the basic characteristics and identifies limitations of the exercise setup. Those constraints have to be considered during evaluation of the conducted projects with regards to their possible influence on observations made during such case studies.

2.1. Characteristics

The foundation for the testbed is a third year undergraduate software engineering lecture. Due to the restricted number of students that our institute admits each year, every class consists of approximately 80 students. Participation in the class and the corresponding exercise is mandatory. Accordingly, the results are less likely to be biased by an artificially increased degree of motivation and skill that voluntary participants usually show [8]. While this setup ensures a comparable standard of knowledge with regards to university education, previous working and project experience may vary and need to be assessed upfront.

Project duration is limited by the fact that the project is part of the exercise that accompanies the lecture. It is being held during the winter term, which usually spans 18 weeks and, as a special obstruction, includes the Christmas holidays. Consequently, the effective working timespan can be expected to last approximately thirteen weeks.

As previously mentioned, prior working experience as well as knowledge of the problem domain and the used programming languages or environments cannot be controlled. Therefore, the project plan accounts for a two week period of initial training before the beginning of the actual development phases.

Students participating in the course are close to finishing their undergraduate studies. They are required to participate in so-called bachelor projects. Each project team is assigned its own working space with adequate technical equipment. Thus, problems regarding team internal scheduling or finding suitable working areas do not have to be solved during project runtime. Additionally, bachelor project teams are required to consist of four to eight members and, hence, ensure optimal team sizes for effective collaboration [6]. Students are assigned to the teams by university administration based on preference lists. This leads to teams that are comprised of students with similar fields of interest and reduces the possibility of teams solely consisting of students that are close friends with each other.

2.2. Constraints

Dedicated workspaces and the team composition are important factors since they closely resemble conditions in an industry setup. Other factors, however, cannot properly be reenacted. First of all, it is not feasible to provide the students with a salary. Grades
serve as a substitute for this kind of compensation, but it is not proven that their impact on the extrinsic motivation is equal.

Secondly, the project is missing the pressure of having to ship to a certain target market within a predefined amount of time and with a required minimal set of functionality. Even though the project duration is limited by the time constraints of the university term, the impact of a failing project is hard to compare with the implications it would have in a real life scenario. On the other hand, grades are tied to project success and a set of external partners is present and has genuine interest in the project outcome. This might result in equal pressure to succeed.

Working time is also a constraint of the study. University rules limit the time spent on the project to eight hours per week. This induces periods of vocational adjustment on a weekly basis - an effect that would not occur in a full-time project.

With the given characteristics, the testbed enables the application of a variety of different process models. The preset sub-teams can be combined in any given fashion, yet still work together efficiently because of the close proximity between their distinct working areas. Additionally, it is possible to split the entire class into multiple projects of smaller size to compare different approaches during the same term.

3. Observation methodology

As previously mentioned, case studies conducted in our project setup aim at analyzing team interaction and collaboration and identifying possible indicators for disadvantageous trends within the collaboration behavior of software development teams. Therefore, it is crucial that proper means are installed to capture a broad range of valuable information about the participating students and as much aspects of their collaboration throughout the project as possible. Of course, data privacy laws are respected at all times.

3.1. Team observation

At the beginning of the project, the students are asked to complete a questionnaire about prior domain knowledge and experience with the programming language and development framework. This information is not only helpful for later evaluation of the team performance, but also allows for on-the-fly adaptations of content and extent of the initial preparation exercise.

During the project, the teams are monitored by graduate level student tutors. Weekly meetings, where the students inform the tutors about their individual responsibilities and the overall team progress, are mandatory. The tutors are instructed to complete questionnaires after each meeting but have the possibility to freely add additional comments.

Based on the chosen development process, the teams perform various meetings (e.g., sprint planning, review and retrospective meetings) with the tutors, where they are supposed to reflect on their teamwork, as well as the collaboration with other teams within the project. The tutors are instructed and trained to moderate those meetings unobtrusively and capture the opinions of the students without further interference. To account for the fact that students might not have the courage to criticize their teammates, the tutors, or the teaching staff in such a setting, they additionally have access to an anonymous email account. By that, they can highlight important negative developments within the project team without having to fear any reprisals by their colleagues or the teaching staff.

At the end of the project, the students are asked to repeat the introductory survey about their domain knowledge and programming environment experience. Furthermore, this final questionnaire includes questions about their teamwork, the way they have experienced the project, and a variety of free text fields for general remarks about each of those aspects.

3.2. Computational analysis

To assist the work of the tutors and compare digital collaboration traces of the participants with their manual observations, means for the computational analysis of team collaboration artifacts are used within the experiment. It has to be noted, that this data only covers a small portion of actual collaboration activity and, thus, delivers merely circumstantial evidence. While it has been shown that certain patterns might indicate problems within the project [43], the results of the computational analysis should only serve as starting points for further, manual investigations.

Technical foundation for this analysis is the d.store platform [44]. It uses Semantic Web technologies to represent concepts of collaboration artifacts, such as emails or wiki pages, as ontologies. The concepts are linked through associations, e.g. a person is linked to an email by being its sender. The concepts are time-annotated and, accordingly, can be put in correlation with the project timeline.

The original implementation of the platform provided support for emails, wiki pages, and WebDav [36] folders. Thus, in order to account for the special requirements of software engineering projects, ontologies for the domains of source code management...
systems and bug tracking items were created (see Fig. 1). With those extensions, we are able to collect all publicly available collaboration artifacts of the project at a single point.

![Diagram showing various tools and services including Email Archives, Wiki, Logs, Source Code Management System, Revisions, d.store, Team Communication Networks, Graph Analysis, Researcher, Developer, Bug Tracker, and Tickets.]

Figure 1: Extensions to the d.store service-landscape.

Aggregating this data into a single team collaboration network allows for simpler detection of uncommon deviations from usual patterns and helps relating them to other events that might have occurred. If, for example, certain teams suddenly utilize the source code management system much more frequently than before, the tutors could easily find out if this sudden boost in activity was preceded by new tickets, emails from the teaching staff, or maybe the creation of a wiki page that contains recent user research results.

3.3. Performance evaluation

In order to compare certain approaches to collaboration, success metrics need to be determined. As hinted by the variety of proposed rating systems (e.g., [39], [28], [45]) this is neither a trivial task nor one that has only a single correct answer. The educational background of the project further complicates the determination of a success metric because learning experience for the students has to be ensured, as well.

During the first iteration of the course, focus resided on testing the setup of the lecture and the observation techniques under real conditions. Thus, success metrics were hard to define or became obsolete due to unanticipated workarounds by the students. In the next iterations, however, metrics will be defined that allow to us to quantify the positive or negative effects of certain tools or techniques and might help us to identify patterns within the digital collaboration traces of the teams that indicate possible detrimental developments.

4. Initial case study

During the course of the winter term 2009/2010, the first case study was conducted using the previously described testbed. Main goals were testing a possible team setup and infrastructure, as well as verifying and optimizing some parameters of the testbed itself. Especially tools like the questionnaires and the computational analysis platform had to be examined during the course of a real project in order to identify ambiguities or unforeseen problems.

4.1. Project outline

The task of the project was the implementation of an enterprise resource planning (ERP) system targeted at small startup companies. Their needs are very diverse, thus the system had to include the most common modules of ERP systems: Financials, Human Resource Management, Customer Relationship Management, and Reporting. Additionally, the system had to provide means for role management, authentication, configuration, and connectivity to external systems.

Three real startup companies served as potential customers for the software. The problem domain was unknown to the majority of the students as their previous studies did not include a lecture about this topic. This domain was chosen deliberately, since it ensured that the students were not able to validate their implementation by themselves, but forced to perform user research interviews with domain experts.

4.2. Development process structure

Main ideas of Scrum [33] were used as the foundation for the team setup and infrastructure of the case study, which are presented by the FMC [21] diagram in Fig. 2. The 78 students of the project formed 13 sub-teams that conducted their work over the course of four equally long (i.e., approximately three weeks) sprint cycles. Each team chose a Scrum Master that was responsible for moderating the weekly Scrum Meetings with the tutors as well as for controlling the collaboration with other sub-teams.

Six students were chosen as Product Owners (POs). Each PO was assigned to two teams, one being the team of his own bachelor project. One PO was required to be responsible for a third team due to the odd total number of sub-teams.

The main responsibility of the PO team was determining the requirements for the system by performing on-site user research at the aforementioned startup companies. The gathered requirements subsequently
were split up into 13 topics and broken down into various user stories. Upon the start of each sprint, each PO presented the prioritized list of user stories to the teams. During this sprint planning meeting, the teams estimated the required effort for each story using “planning poker”. By that, they also decided which stories were considered for development during the upcoming sprint. At the end of the sprint, POs and teams met again for a sprint review meeting to assess, which requirements were successfully implemented and which needed to be carried further into the next sprint.

The students decided to conduct a weekly meeting of the Scrum Masters, or other representatives of each team, to solve the following problems:

- coordinating the collaboration between teams that implemented mutually dependent requirements,
- creating an institution that was able to solve problems, which affected the entire project,
- and maintaining an overview about the other teams’ current work in progress.

This so-called “Scrum-of-Scrums” was performed in a fashion similar to the one used in the weekly scrum meetings with the tutors.

In addition to the high-level user research conducted by the POs, the teams were required to validate their implementations with fictional stakeholders that were impersonated by members of our research group. The point in time, at which the interviews should take place, was not defined. Most teams performed their user research interviews either at the end of the third, or during the course of the fourth sprint. Furthermore, the teams had to perform at least one single code review and refactoring session with a programming language and framework expert from our teaching staff. Those meetings almost exclusively took place during the last two sprints, too.

### 4.3. Technology Setup

The web-application development framework Ruby on Rails [14] was used as the technological foundation for the project. The framework was chosen for two reasons: Firstly, dynamically-typed languages vastly reduce turn-around and implementation times by offering a programming paradigm that embraces change of existing implementations [42]. This makes them a natural choice for agile software development processes. Secondly, the framework is supported by a strong community that frequently produces useful plugins for many aspects of software engineering tasks.

The Cucumber framework [15], for example, facilitates principles of Behavior Driven Design (BDD) [26] by allowing the POs to define their user stories in an easily comprehensible domain specific language and automatically use those descriptions as integration tests for the system. RSpec [11] was used to simplify unit testing for the students and intensify the learning experience regarding Test Driven Development (TDD) [13].

To further aid the development process on an infrastructure level, a CruiseControl [18] server was set up for continuous integration (CI). Furthermore, we provided the students with a shared Subversion repository [29] and CodeBeamer [37], an application lifecycle management tool that features wiki functionality and a bug tracking system. Distinct mailing lists for each team, the POs, the Scrum Masters, and the teaching staff were created, as well.

### 4.4. Analysis of digital collaboration traces

As stated earlier, the main focus of our research was the analysis of the digital collaboration traces created by the students during the course of the project. The following artifacts have been created:

- 2488 source code revisions,
- 1068 emails (excluding messages from the CI system),
- 303 tickets (with a total of 1048 status changes),
- 92 wiki pages.

They were uploaded into the d.store and, in turn, became part of the overall collaboration network that reflects the collaboration behavior of the development
teams with regards to digital tool usage. The platform offers a very flexible SPARQL [30] query interface that fosters the fine-grained analysis of the data at hand. Queries can range from simple detection of all artifacts created within a certain time period to complex constructs that try to relate artifacts of multiple types with each other (e.g., querying for all emails written by a member of group X within a two day timeframe after either a revision or a wiki page has been created by members of group Y).

It has to be noted that the following statements cannot be generalized and have no direct, statistically significant correlation to project success. The purpose of this discussion is to demonstrate that digital collaboration artifacts can provide indicators for certain developments within the project teams, that otherwise could only be detected in an extensive manual observation process.

Most students mentioned that Subversion, was a major source for problems that interfered with the progress of the project. Since the sprints of the teams ended on the same dates and the project only used a single trunk in addition to the 13 distinct branches created for the teams, merging was often the source for conflicts. Data analysis revealed that over 95% of all commits to the trunk that contained the word “merge” happened in a timespan of two days around the end of the sprints and 64.8% of those commits resulted in a failing run of the CI server. During the meetings with the students, we tried to analyze those situations and it became apparent that bridges to decentralized source code management systems (e.g. Git [10]) for team-internal development caused these issues. Additionally, many students felt insufficiently prepared to work with Subversion, even though a distinct tutor for that topic was available.

While monitoring the activity in the wiki and ticket systems, we noticed that especially changes to the tick-

![Figure 3: Number of ticket changes and source code revisions per day.](image-url)

ets only occurred in clearly visible time clusters (see Fig. 3). Status changes, such as opening, closing, or updating a ticket with the latest development progress happened mainly near the start and end dates of sprints. Since the ticket system is meant to reflect the current state of certain implementation, this could indicate that either development was not continuous or the tickets were simply not updated on a regular basis. By adding the dimension of source code revisions to the data analysis, the latter assumption can be explored further.

Fig. 3 shows that, in the beginning, the curves align on most occasions, with the exception of some bursts in the revision count and a constant deviation. Both phenomena are results of the aforementioned problems with Subversion and reflect merge revisions or fixes of merge errors. However, it becomes apparent that the further the project continues, the less the graphs align. Asked about this particular pattern, students indicated that near the end of the project, they did not pay close attention to the ticket system anymore, but were only focused on delivering the required functionality in time.

Still, this does not explain the clearly visible clustering in the beginning of the project. When asked about the reasons for this behavior, teams and POs alike stated that the system is too cumbersome to use and, hence, they only updated the tickets in a bulk procedure. This often resulted in a discrepancy between the state of the tickets and the actual development progress. Hence, POs and the teaching team had to find other means to obtain this information. Some statements in the final questionnaire indicated that a simple and easy to use bug tracking system would have been utilized much more frequently. This assumption will be tested in the next iteration of the lecture.

Those two small examples show that the collected data is a viable means to draw conclusions about the collaboration behavior of the team and can be used as a starting point for manual corrective interventions. Interestingly, a lack of data is also an indicator for possibly overlooked aspects of team collaboration. The relatively low amount of emails generated by the teams - less than one per day per team member - was also subject to further investigation. It turned out that most communication was performed through other channels, such as Facebook, IRC, telephone, or in direct conversations.

Summarizing, it can be noted that this initial experiment was insightful with regards to the possibilities that the analysis of digital collaboration traces offer, as well as with indicating future extension points and improvements for the platform.
4.5. General project feedback

As presented in Section 3, an intensive retrospective process has been performed throughout the entire project. The following section presents student feedback gathered from notes taken during retrospection meetings, a special event where all 13 teams shared their experiences, and the concluding questionnaire. While, due to a lack of success metrics, those opinions do not provide statistical evidence for positive or negative effects of certain aspects of the development process, they provide hints about topics that might need further investigation in future iterations of the lecture.

Regarding the development process itself and the collaboration between the teams, the feedback was very diverse and ranged from very positive opinions about this particular version of Scrum to the request that a completely different process model needs to be established. One of the main problems stated by numerous participants was the lack of coordination between the different sub-teams. The Scrum-of-Scrums, which was specifically installed to solve those problems, was only considered to be helpful in this regard by 42.8% of the students. Another 42.9% were unsure whether it was helpful or not. This, and the comments from the direct conversations, leads us to think that (1) these meeting were conducted inefficiently and (2) the Scrum Masters had problems to communicate the results of those meetings to their teams.

Another issue that was mentioned during 11 of 13 final retrospective meetings was the missing overview about the system itself and a only rudimentary understanding of the dependencies between different parts of the system. One student briefly described this problem by mentioning: “We only saw our user stories and had no idea what other stories led to our functionality.”. The general opinion of the students was that the only group of people that had an overview about the system’s parts and their intended interplay were the POs. However, as they were forced to break down new requirements into small, assessable user stories, their knowledge was obviously not properly transferred to the development teams. This led to some sort of bureaucratic mentality (stated by 57.7% of the survey participants), i.e., many teams only performed exactly what was specified on the user story cards or covered by the corresponding Cucumber tests and did not tackle obvious issues, e.g. related to usability.

Despite the fact that only 30% of the students experienced an equally distributed workload within their teams, over 80% got along really well with their teammates. So, obviously, even amongst students, a good working atmosphere is no guarantee for fair sharing of workload and evenly distributed responsibilities. Statements made in the free text area of the questionnaire are backing this thesis, as students complain about other team members pulling back from teamwork due to insufficient means to track individual performance.

As a last point, the user research interviews, that the teams themselves carried out, were also a topic of the retrospective. They were sought to be helpful in understanding the problem domain much better (71.4%). However, as previously mentioned, the students were free to choose the point in time at which to conduct those interviews. During the meetings, it was often stated that the teams were unsure whether to perform such interviews with only partial solutions. Accordingly, 32.1% could not decide if the moment for user research was a good choice and 46.4% thought that the date was poorly chosen. It was also often mentioned that only during those interviews the teams were confronted with workflows that did not work as expected by the users. Obviously, the developers either had no idea about how proper interaction patterns should look like, or they were not willing to fix errors they might have recognized during the development. Either way, those observations led us to the conclusion that a distinct user interaction design team might help to improve the usability of the overall product. Accordingly, in the next iteration of the lecture, such a team might be installed.

Despite the problems revealed during the project, its educational effects were very well perceived by the students. 71.4% of the students agreed to the statement: “The project gave me an impression of developing in a large project team”. Also, in a list of the top five things that should be repeated next year, over 50% of the students included the project size and the basic organizational structure that makes use of the bachelor project groups as development teams.

5. Related work

Experimentation in quasi or controlled experiments is a widely adopted approach in software engineering research [35] but also considered to be a viable solution for the education of computer science students [38].

Allen et. al [1] performed an open source project within on of their software engineering lectures. It included many aspects that our project comprises, such as user research, test driven development, and tool-aided project management by a team of project managers. The project manager team is comparable to the PO team of our case study. In contrast to our approach, the students started with an already existing open
source project and the project followed strict Extreme Programming (XP) methodology [6]. Furthermore, the students did not work on a single, large project, but three, mainly independent, assignments to extend the functionality of the given system.

Another experiment that deals with large-scale projects in educational environments was conducted by Tan and Teo [41]. They equally focussed on real-life problems, for example a human resource management system, and encouraged requirements engineering practices performed by the students by providing partners from industry. However, their focus resided on improvements in the creation and identification of innovative functionality in the overall product instead of optimizing the workflows within the development processes of the projects themselves.

Bowyer and Hughes [9] investigated the effects of test driven development and CI technology on the development process in a classroom project. While many elements of the study are comparable, for example the initial preparation exercises or the time constraints, their experiment was limited to determining the effects that CI has on programming pairs that not have been part of larger development teams. Despite this restriction, their results were insightful and helped us during the preparation of the environment for our project.

Besides the project and the corresponding development process assessment, the observation of digital collaboration data was also an important focus of our work. An interesting study covering a similar topic has been performed by Edwards et. al [12]. They installed a system that was able to give students instant feedback about how well they have tested their programs and, in turn, improved their programming performance. While our system is capable of the same evaluations, we deliberately decided to not make those results available to the students, but only the tutors, in order to avoid optimizations based on recognizable grading patterns.

Another approach that aggregates collaboration data from different data sources has been developed by Ohira et. al [27]. The Empirical Project Monitor relies on numerous feeder applications that parse datasources like source code management systems, bug trackers, or email archives. It provides a number of preset visualizations for this data. Additionally, an underlying communication model tries to detect flaws within the collaboration behavior based on empirical studies.

Reiner [31] presented a proposal for a knowledge modeling framework that supports the collaboration of design teams. Communication information was deduced from explicit interactions between members of a design team by a software tool that he developed to provide a prototypical implementation for the proposed framework.

While these two approaches are able to create the same views on the available communication artifacts, the semantically annotated models of the d.store can be easily extended with new domains by adding the respective resource description to the application. Additionally, external services that parse the information available in various sources can feed the d.store networks with new data. This also allows to simply re-use existing resource definitions for the information from different implementations of the same concepts (e.g., different wiki types). The aforementioned implementations would require extensive adoptions of internal models to achieve the same behavior and defining arbitrary queries on this data would require knowledge about those internal data structures.

6. Summary and future work

In this paper we have presented a novel approach to the testability and evaluation of collaboration behavior in early stages of software engineering processes. Our approach defines a controlled educational environment that allows to perform software development projects with realistic team sizes and using differing problem domains, technologies, and adoptable process models. We furthermore presented the means used to observe the process both on a personal as well as on a technical level and discussed the insights gained during an initial case study. Finally, we examined related work in the field of software engineering process experimentation.

Our future work is mostly based on the insights gained from the first case study as it revealed a variety of possibilities for improvements of multiple aspects of our research.

Regarding the tools for computational analysis, especially means to visualize the data efficiently need to be created. As mentioned earlier, the aggregation of all collaboration artifacts allows the tutors to gain deeper insights into possible problems of teamwork. With this knowledge, they are able to perform their tasks much more target-oriented and interfere only if indicators exist for certain problems. However, the current user interface is very technical and does not provide them with a single-screen, dashboard-like overview about current developments within the teams they are responsible for.

The chosen process model for the project has also unveiled points that could be enhanced in future iterations. Especially the missing connection between the functional units of the software could be traced down to an excessive separation of concerns without a connecting team whose sole responsibility is ensuring
the implementation of proper workflows. This fact, and some other findings have led us to creating a new, improved process model that will be tested during the next iteration of the lecture.

As soon as the platform for the computational analysis has been extended and optimized in the proper ways, the next step is its provision as a service to other researchers. Our university is only able to conduct such a large project once a year. Accordingly, we can only slowly progress in terms of process improvements and further insights based on the computational observation. Therefore, it will be beneficial to provide other universities with the possibility to use the d.store platform, as well. A software-as-a-service platform will enable tutors of other institutions to reason about their projects, too, and, by sharing their assessments, help to improve development process models by identifying best practices and the according patterns within the digital collaboration networks that indicate them.

References


