AnalyzeD: A Shared Tool for Analyzing Virtual Team Collaboration in Classroom Software Engineering Projects

T. Kowark and H. Plattner
Hasso Plattner Institute for IT Systems Engineering
University of Potsdam
Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract—Classroom software engineering projects are viable aids for the education of computer science students. They allow to experience different technologies, methods, and development processes first hand, and intensify the learning effect compared to traditional lectures. But how to track whether the learning goals of such projects are fulfilled? Main focus in project evaluation often resides on results, since tracking the process itself can only be achieved with substantial manual efforts.

We present AnalyzeD, a tool that simplifies monitoring of development processes within such settings. It enables observation of groupware systems for the occurrence of behavior that reflects defined learning goals like adherence to principles of the taught development processes or effective and efficient usage of the provided tools. More importantly, it allows sharing analysis results, thereby, breaking up the isolation of different classroom projects and allowing all users to benefit from previous insights into the inner workings of student teams.

Keywords: Collaborative software, Computer aided instruction

1. Introduction

Classroom software engineering projects have a long standing track record in aiding computer science education by enabling students to experience the theoretical foundations taught in software engineering lectures first hand in risk-free environments [2], [4], [19], [20]. Classroom projects, not only limited to the domain of software engineering, have also been a viable source for scientific insights into the inner workings of student teams.

Collaboration is nowadays increasingly focused on digital tools. Groupware systems, such as email lists, wikis, bug tracking systems, or version control systems, are ubiquitous in modern software engineering projects and much effort is put into connecting the remaining analog artifacts of software engineering projects to their digital counterparts. Analysis of these digital artifacts has shown to provide indicators for the current state of engineering teams and even the expected project outcome [21], [22].

Based on these findings, we developed AnalyzeD, a system that allows to capture, analyze, and compare such collaboration data from arbitrary project settings and groupware landscapes. Virtual collaboration artifacts in that regard are traceable actions within the provided systems, e.g., editing of a wiki page or creation of a new source code revision. The system further allows to share the results of the analysis of virtual collaboration artifacts directly with other users, which, in turn, can try to reproduce the observations within their respective project settings. Thereby, AnalyzeD extends the available database for verification or falsification of theories about the impact of certain virtual collaboration behavior on collaborative work beyond single institutions.

Such a system is especially interesting in classroom settings, since the teaching team to student ratio usually prohibits to observe each student or student team as closely as it would be necessary in order to constantly recognize deviations from the intended development processes or advised usage patterns for provided groupware systems. By providing a unified view on collaboration activities in all accessible collaboration tools, indicators for such deviations and misuses can be recognized and permit to provide more focused feedback to the students. Beyond that, the system offers the possibility to specify desired and unwanted behavior and share it with other users of AnalyzeD. This means, if indicators for good or bad project work have been identified, the entire user group of the tool immediately has access to this information and can search their own collaboration activities for occurrences of the described behavior.

This paper intends to make AnalyzeD known to a wider audience of software engineering lecturers and encourage them to use it within their educational software engineering projects or experiments. The remainder of the paper is structured as follows. First, we give an overview about related work in the field of virtual collaboration analysis. Then, we provide a detailed description of the system architecture. Finally, we present experiences made during a case study in a software engineering lecture. The paper concludes with an outline of next steps in the development of the system as well as further research work that can build upon the presented efforts.
2. Related Work

The idea of analyzing digital artifacts of team collaboration in software engineering projects was subject to prior research. The tools presented in related work, however, either lack the flexibility to be applied in arbitrary groupware setups or were isolated solutions that did not facilitate publication of analysis results to other users. In the following, we present an excerpt of publications about such tools and outline the similarities and differences to AnalyzeD.

Hackystat is an open-source tool that aims to automatically and unobtrusively collect, analyze, and interpret software development process and product measurements [10]. It allows to capture usage data from a variety of development tools and analyze this data through interchangeable client interfaces. The implementation is not designed to facilitate sharing of collaboration traces and analysis results since different installations remain isolated from each other. With regards to specifying desired or unwanted behavior, extensions of the platform were, for example, used to automatically determine whether programmers adhere to the principles of Test-Driven-Development (TDD) [9]. The efforts presented in this paper are a generalization of such extensions since they support not only singular use-cases within closed environments, but facilitate the encoding of collaboration behavior utilizing any possible combination of collaboration tools.

Microsoft’s Team Foundation Server [5] is a collaboration tool suite that allows its users to analyze and compare their own collaboration behavior with other teams that use this platform. It is, however, limited to collaboration activities that are performed using the platform itself. If different version control, bug tracking, or email systems are used, the corresponding activities cannot be analyzed.

The Empirical Project Monitor is a tool that aggregates collaboration artifacts from different data sources, as well [14]. Similar to AnalyzeD, it relies on feeder applications that parse data sources like source code management systems, bug trackers, or email archives. A communication model is provided in order to detect potential problems within the collaboration behavior based on empirical studies. However, this model is encoded within the software itself and cannot be adapted to incorporate new knowledge without recording and redeploying the system itself.

Wu et al. created a metric-based, multi-agent system that supports project management by gathering information from various sources of the development environment and deducing the current project status by means of intelligent agents that relate software development activities to the previously created project plan [23]. The system is applicable to various project setups, but the different installations remain isolated from one another. Insights about activities, which have effects on team performance cannot be made available to other projects without updating the evaluation logic of the software agents and redeploying the system.

A slightly different approach was taken by Reid et. al [16]. They introduced DrProject, an entire platform of project management tools that was specifically tailored to educational needs. By developing the entire groupware landscape by themselves, they simplified the process of collecting and analyzing virtual collaboration activities in order to better understand how students work together in classroom software engineering settings. This simplicity in data collection comes at the cost of realism with regards to the tools used and reduces generalizability of the analysis results since no abstract model of the collaboration activities is provided.

3. System Architecture

AnalyzeD is based on a previously developed platform for virtual collaboration monitoring [13]. This platform was rebuilt to be available as an on-premise solution, which can be unobtrusively deployed alongside the infrastructure used for the projects under investigation. Further, it was extended with a central repository that allows to publish the results of virtual collaboration analysis and make them available to other users of the system.

3.1 Local Platform for Data Capturing and Analysis

The local analysis component of AnalyzeD is presented in detail in Fig. 1. It consists of three components. Firstly, specialized sensor clients capture collaboration events from the groupware landscape. For each used groupware tool, a specialized sensor client needs to be available. Regardless of the parsed systems, each client needs to implement an interface that allows the “Configuration and Control Service” to get basic information about required input parameters, to invoke the client, and to retrieve the results, i.e., the parsed collaboration events, in a common format. Thus, the system remains adaptable to newly created groupware tools.

![Fig. 1: Detailed Architecture of the local analysis platform component of AnalyzeD.](image-url)

The data returned by the sensor clients is being aggregated into so-called Team Collaboration Networks (TCN) [21].
These networks are built upon ontologies that represent concepts of different collaboration tools, such as wikis, email lists, or version control systems. Thereby, the system can easily be extended just by creating a new ontology if new tools for collaboration are created that differ conceptually from the tools we know today. All nodes of the TCN can be linked by relations. Hence, these networks provide a holistic view that, for example, can model links from revisions to bug tracking items or from emails to wiki pages and allows to analyze whether the occurrence of such links might affect team collaboration. The TCN are stored within an off-the-shelf graph database.

The ontology-based system allows to add additional information to the TCN without having to change underlying data structures. If needed, new concepts or additions to existing ones can be modeled by means of an ontology, which then needs to be uploaded to the local analysis component. After that, sensor clients can be used to upload the corresponding data, e.g. calendar events, questionnaires filled out by project participants, or assessments of group or individual performance, and integrate it into the TCN. While currently not implemented, upcoming versions of the shared repository will allow users to upload these extension ontologies, too.

Analysis of the TCN can be performed by utilizing a SPARQL [15] endpoint, which is offered by the underlying graph database, or by using RDF/OWL compatible visualization and analysis tools.

### 3.2 Shared Repository for Virtual Collaboration Behavior

Analysis of said networks usually reveals reoccurring subgraphs that either have a direct impact on team performance or at least provide indicators for ongoing beneficial or detrimental collaboration behavior within the observed project teams. In [11], we formally described how to model such abstract subgraphs. In addition to the formal definition, we provided a graphical notation that allows to specify said behavioral snippets in a more accessible fashion.

Fig. 2 presents an example of this graphical notation containing all possible building blocks. Nodes are represented as circles that are labeled with possible classes of the node. The label can be extended by a set of valid tags for a node, which is depicted in squared brackets. Relations between nodes are represented by solid, directed edges. The label of an edge denotes the name of the relation and optionally defines cardinality constraints. Sequences can be expressed by dashed edges that connect two relations. Finally, attribute constraints are expressed within a separate rectangle that is attached to the respective node. Attribute constraints can be defined using SPARQL syntax.

As depicted in the architecture overview (see Fig. 3), we created a graphical editor (“Visual Creator”) for this notation. Thereby, users of the systems are able to model behavioral snippets that represent, for example, learning goals within their classes. The graphical notations are translated into SPARQL queries for the purpose of detecting occurrences of the described behavior within the TCN under investigation. The “Matcher” component thereby takes into account characteristics of the respective networks and adopts cardinalities or terminology of tags. Finally, the “Observer” component allows to constantly monitor TCN for such occurrences and notify the user about newly detected instances. In combination, these components can serve as a freely configurable project management dashboard that simplifies the detection of indicators for desired or unwanted behavior within the monitored projects.

The content of the shared repository is available to each local analysis platform. Hence, all behavioral snippets are available for the analysis of local team collaboration networks and statistics about the detection of such behavior will be transferred back to the repository. Thereby, it becomes possible to determine whether observations are specific only to single projects or can be reproduced in different settings, as well.

### 4. Platform Application

In the following, we present experiences of using AnalyzeD within a classroom software engineering project. The lecture under investigation is a third year undergraduate software engineering class [12]. The 96 participants of the course were divided into two development groups, each consisting of eight teams. Those eight teams had the task of jointly developing a customer relationship management system (CRM).

At the beginning of the project, the students had to perform a mandatory preparation exercise regarding the
used programming language and frameworks. They received tutorials for the groupware infrastructure, which consisted of dedicated email lists, Agilo [7] as a wiki and bug tracking solution, GIT [3] for version control management and Hudson CI [8] for continuous integration. The project itself was performed using Scrum [17]. Each of the eight teams consisted of four to eight members, one of which took the role of the Product Owner (PO) and one served as the Scrum Master (SM). The POs had access to a team of customers embodied by members of our research group and performed initial requirements elicitation one week prior to project start. After that, all teams performed four sprints. Each sprint started with a sprint planning meeting and ended with a sprint review and a retrospection meeting. During sprint execution, the teams performed weekly Scrum meetings where they discussed ongoing developments and presented their work to the teaching team. Each sprint lasted three weeks and working time was officially limited to one day per week, however, additional work was not prohibited. The students were not allowed to freely choose their project teams but had to work with their colleagues from their concurrent capstone projects. By that, it was ensured that all teams could use dedicated rooms and workstations for the project.

The students received regular lectures that provided the theoretical background for the project work. Topics included different approaches for scaling Scrum to multiple teams working together in one project (e.g., [1]), introduction into development methods, such as Test- and Behavior-Driven-Development, and detailed tutorials for the tools used within the project.

From an educational point of view, the main focus did not reside on the overall project outcome, but on the adherence to the prescribed development process and usage of the presented tools in a way that contributes to project success. Therefore, we installed a team of four tutors that constantly monitored the progress of the teams and determined whether they adhere to the principles of Scrum and, if they do not, investigated the reasons for that. The tutors took part as silent observers in all meetings of the teams - weekly Scrum, sprint planning, sprint review, retrospection, SM, and PO meetings - and recorded their observations.

The presented iteration of the course is the second one. During the first iteration, we learned that manual observation during the meetings was not sufficient to completely capture the students teamwork. Firstly, teams avoided to mention problems in front of the teaching staff to prevent negative impacts on their grades. After the grades were announced, one team revealed that they thought co-located working would be rated higher than each team member performing their tasks individually. Hence, they tried to cover-up that one of their members was constantly working alone at home instead of the dedicated project space by excusing him during some regular meetings. Secondly, the tutors sometimes forgot to ask important questions and teams forgot to mention certain actions they took to improve their project work. For example, one project team was taking daily photos of their Scrum board and uploaded them into the wiki instead of using the provided ticket system. The tutor became aware of this fact only two weeks after the team started to exploit that behavior.

As a consequence of these examples, we decided to use
AnalyzeD as an additional tool for analyzing the virtual collaboration traces, which were created during the work that was carried out in between the scheduled meetings. The tutors, however, did not have access to the system. By that, we tried to ensure that they neither intentionally nor subconsciously tried to push the teams towards creating expected behavioral signatures.

4.1 Digital vs. Analog - Bug Tracking System Analysis

Being an agile software development process, Scrum does not focus on tools but on individuals and interactions. Hence, Scrum teams are supposed to find the most suitable tools for their work instead of dogmatically adhering to existing systems. Within the lecture, one intention was to teach the students that, even though digital bug trackers are viable aids within the software engineering process, they are sometimes inferior to analog solutions, such as physical Scrum boards.

The following observation was made during post-hoc analysis of the first iteration of the course: when teams started to focus on physical Scrum boards for ticket maintenance, the digital versions of the tickets were increasingly updated in bulk by designated members of the development teams. This permits to maintain an overview about the current progress directly within the team office, while simultaneously allowing to share the progress with other teams and the teaching staff through the groupware system with minimal effort. In order to detect this style of working, two behaviors were modeled.

Fig. 4 depicts that a ticket is changed by its owner. The owner in that regard is not the person that created the ticket, which often would be the PO, but the person who was assigned to implement the described feature or perform the desired task. We assumed that such behavior would mostly be exploited at the beginning of the project and decrease over time when small changes are carried out exclusively on the physical Scrum board.

Fig. 4: Behavior 1. A ticket is changed by the developer that owns it.

The second behavior, updates by other team members that do not own the ticket, is depicted in Fig. 5. It is assumed to increase during the project as more changes are performed on the physical Scrum board and designated team members start to update the digital tickets every couple of days for their teammates.

We used AnalyzeD to model these behaviors and monitor their occurrences. Fig. 6 shows the detected number of occurrences during the project for an entire development group. It is clearly visible that, overall, the assumptions about the occurrence rate of the behaviors held true. After many developers initially edited their own tickets on a frequent basis, turnaround was reached midway through the project, where changes by non-owners became more frequent.

Fig. 5: Behavior 2. A ticket is changed by a team member that is not the owner of the ticket.

Fig. 6: Development of occurrences of the behaviors under investigation over the course of the project.

In addition to a graphical indicator for reached learning goals, the observation mechanism had a second positive aspect to it. By narrowing down the number of tickets worth investigating to just a fraction of the overall activity in the ticket system, i.e., those that were changed by their owners, it allowed us to provide more focused feedback to the students compared to just giving general advices to the entire class of 96 people. Resuming the example above, where the team tried to cover-up the student that refused to work with the entire group, AnalyzeD helped us to detect such behavior during project runtime instead of only in post-project evaluations and, thus, allowed us to react accordingly. Tutors could investigate the reasons for the behavior and tried to find suitable solutions in consultation with the teams.

4.2 Limitations

Digital collaboration is just a fraction of overall collaboration in software engineering projects. Therefore, the analysis of this data can only assist manual process observation, not
replace it entirely. When applying the system, some aspects need to be taken in consideration to meaningfully analyze the collected data.

Connected tools can lead to false conclusions. If, for example, the bug tracking system is connected to the version control system by detecting ticket references in commit messages and automatically performing the corresponding ticket changes, focusing on only the ticket system during the analysis would be insufficient. Instead, it is then necessary to extend the respective ontologies in order to reflect the link between the two systems. Further, the behaviors modeled above need to be adopted to that situation and have to include source code revisions to detect tickets that were changed by their respective owners by referencing them in commit messages.

Another factor, which needs to be taken into account, are usage mistakes. Especially at the beginning of projects, students tend to misuse groupware tools due to inexperience. This could lead to frequent ticket updates without meaningful changes or a high number of source code revisions that fix errors of previous ones. In the case study, however, these unexpected usage patterns revealed that the tutorials for some of the tools were insufficient and additional training was necessary.

Finally, digital collaboration analysis is prone to observer effects. If students know that the way they use groupware tools is analyzed, it becomes much more likely that they adapt their behavior to generate supposedly ideal traces instead of focusing on doing what is best for the project. The students in our case study were informed upfront that their tool usage is being monitored and how we expect them to use the tools. Still, it was possible for us to detect the presented examples. While obviously not all students optimized their usage behavior to comply with our expectations, we cannot neglect the possibility that an unknown number of students did just that. Therefore, virtual collaboration analysis should only accompany thorough manual guidance and the results must always be put in context to the respective project setup.

5. Conclusion and Future Work

In this paper, we presented AnalyzeD, a system that allows to capture and analyze virtual team collaboration activities in an unobtrusive manner and publish the results of the analysis to a central repository that is available for all users of the platform. It is well-suited for classroom settings as it allows comparison of collaboration activity across different groupware setups and promotes the exchange of knowledge about detrimental or beneficial virtual collaboration behavior between software engineering teaching personnel of otherwise unrelated institutions.

The presented application is a simple example that showcases the general idea of the system. Due to the extensibility of the TCN approach and its ability to reflect not only the entirety of visible virtual collaboration activity in projects but add additional meta-information about project participants, more complex examples are likely to emerge.

Use cases that will be investigated in upcoming experiments include: determining whether teams are dealing with the tasks at hand or if they are spending time on unnecessary tasks, detecting hot spot teams or team members that are overloaded with tasks due to extensive dependencies, and verifying if adherence to development principles like Test-Driven-Development can be detected by virtual collaboration analysis and if it is possible to quantify their effect on team productivity and performance. A shared platform that allows to publish such findings and enables other researchers and practitioners to falsify or verify them without the inherent delays of traditional publishing methods promises to be a viable aid for this kind of research.

6. Availability

The system is currently being prepared for open-source release under a yet to be determined license. Additional information and downloads are available at http://epic.hpi.uni-potsdam.de/Home/AnalyzeD.

References


