Analysing group learning in collaborative software development teams

Anonymous
Anonymous Institution
anonymous@anonymous.edu

ABSTRACT

The supervision of collaborative software projects is a great challenge for teachers. All learners involved must be able to participate in the learning process and group collaboration must be ensured, while the program code can take on a considerable size. In this paper, we identify, define, and validate a total of 32 indicators for collaborative learning in software development teams. The resulting model describes collaborative interactions in programming teams considering 7 indicators for code quality, 16 group participation indicators, and 9 indicators for group cohesion. These indicators have been applied to 11 groups of students (N=64) collaborating over a periode of 14 weeks. In addition, we presented a data processing pipeline for extracting, calculating, and visualizing these indicators on a teacher dashboard. This approach enables teachers to keep track of complex group activities and individual contributions, and subsequently provide targeted formative feedback to the groups.

Keywords

Teaching Collaborative Programming, Educational Data Mining, Learning Analytics, Group Assessment

1. INTRODUCTION

Software development takes place almost exclusively in teams, so it is especially important to be a team player, to be organized, and to communicate properly [2, 25]. Learning such competencies is crucial for employability and has great relevance for the job market. In higher education, these skills are therefore often trained in groups, within the context of computer-supported collaborative learning (CSCL) [28]. In computer science education, students must be prepared for this collaboration by developing their competencies in the required methods (e.g. Scrum or Adaptive Software Development), tools, and the programming languages used. Hence version control systems (e.g. Git, CVS, and SVN) and issue tracking systems (e.g. GitLab, Bugzilla, and Zendesk) are widely used, both in education and in the software industry (e.g. [15]). These systems generate data that can be used not only for risk analysis of projects (e.g. [20]) but also to support students working together on a software project.

In this paper, we argue that teachers can benefit from monitoring tools that represent students' contributions in collaborative software development tasks. When supervising student teams in these settings, teachers need to maintain the learning situation for the students, ensure that all team members acquire knowledge in the different domains (e.g. project management, design, development, and testing), support completely heterogeneous students (e.g. regarding skills, pre-knowledge, and interests) when problems and questions arise, and provide them appropriate feedback during the development phases. This guidance and support is an enormous challenge. In software project teams, the lines of code increase enormously over time. The code and its quality can often only be analysed with great effort. Furthermore, the individual contribution of each student across multiple files, commits, and branches is not easily to identify. Hence, the degree of collaboration can only be examined through student reports. Difficulties of individual students or even free-riding effects can remain hidden over a long period of time, so that the participation of all students in the learning process cannot be ensured. Our presented approach aims to support teachers in the challenges mentioned above. The goal of this work is to identify, define, and validate indicators for collaborative learning in software development teams. To address the particularities of collaborative learning and software engineering, we pose the following three research questions: (RQ1) What indicators can be used to describe learner participation in collaborative programming teams? (RQ2) Which indicators provide insights into collaborative software development? (RQ3) What indicators are suitable for capturing code quality in dedicated programming languages? Finally, a fourth research question is posed combining the answers of the previous questions: (RQ4) How can teachers be supported in using these indicators? The indicators are derived from existing literature and adapted to the subject matter. Before the indicators are prepared for use by teachers, a validation is performed using real data sets from four learning groups. As a result of this work, teachers will be able to track complex group activities, code quality and individual contributions, and subsequently provide targeted formative feedback to the groups.

With this paper, we contribute to the field of CSCL and Educational Data Mining. Our contribution to CSCL consists of a model describing collaborative interactions in programming teams considering coding, group participation, and group cohesion. Regarrding Educational Data Mining, we present a processing pipeline for analysing data from version control systems and issue tracking systems. From this pipeline, we gain data for a teacher dashboard for monitoring individual and group-related progress across multiple iterations of software development.

2. RELATED WORKS

In terms of teaching and learning, there is already some work that investigates collaborative software development [4, 23, 26]. However, these only consider individual contributions, not the group collaboration. [14] analyzed commit messages and classified team members as being collaborative, cooperative, or solo-submitters. In contrast to our work, the authors only considered data from the version control system, but did not include the communication and discussion necessary for project management and design. Furthermore, code metrics have not been examined. In a case study [32] investigated the use of GitHub as a teaching tool for individual assignments. By analyzing the commit history and evaluating the comment quality the authors tried to classify students in order to find proxies for grading. This attempt was not very successful. Unlike our research, [32] did not intend to support teaching on collaborative software development through formative but data-driven feedback provided by the teacher. Apart from the commits and the code comments, only a very small set of data was used for the modeling of indicators. The resulting source code and the communication among the students was not considered. More advanced analytics approaches consider, for example, co-editing networks [16], commit quality [1], refactoring detection [31], change patterns [19], and risky commit prediction [24], but without addressing aspects of learning. Beside that, automatic methods are still lacking for specific problems such as common errors in the use of Git [12]. Personal assistance systems support collaboration only on a low-level [9] or even hinder it [34], which is why instructors still play a central role in guiding student groups. In our approach we wanted to use a comprehensive set of indicators to flexibly support different didactic scenarios, team and project structures, and software-technical possibilities.

3. MODEL FOR GROUP LEARNING IN SOFTWARE DEVELOPMENT

Before we can start modeling, we need to know the data that is available for modeling and can later be extracted automatically from the systems used. In this case, we rely on a version control system and an issue tracker. Version control systems provide three types of information: (1) code-related data regarding the quantity and quality of the program (e.g. code smells, security hotspots) as well as (2) logs and (3) content of commits, branches, and merges. Since the content data requires a qualitative analysis in view of the respective task, we focus on quantitatively exploitable data from the source code and the logs which are described in the following subsections. Issue trackers complement this data with created and commented issues, merge requests, and project plans (e.g. Kanban board). The presented model aims at selecting a set of variables from version control system and source code that can be translated into quantitative indicators that are easy to acquire and process by teachers. The indicators should describe individual efforts as well as the cooperation in the team. The chosen subset helps teachers to provide regular individual and group feedback in terms of learning practical skills and increasing employability. Indicators focusing on efficiency gains and risk avoidance appeared to be more relevant in professional and economic contexts and have not been not considered here. The resulting indicators that are relevant for learning can be later used to present an overall picture of the collaborative process within a student-led software development team. This picture is intended to ensure and promote appropriate peer teaching with the focus on supporting students in solving practical tasks or problems and developing programming skills. The model is based on the previous work of [5] on effective group models and the considerations of [18] about teamwork.

3.1 Indicators for software code maintainability (RQ3)

Usually, as the size of a software project increases, so does the number of program errors, security vulnerabilities, and the maintenance effort. Developers therefore try to use tools and selected programming languages to detect certain types of errors at an early stage or even to exclude them completely. Even novice programmers can make use of these tools and improve the quality of their code, as long as they have configured their development environment accordingly and as long as they can understand and implement the advice. The same tools for the analysis of the software code maintainability (cf. [3]) are suitable for the formative analysis and assessment of learning achievements in programming. We consider the indicators S1–S7 in Tab. **??** to be relevant in order to answer RQ 3.

3.2 Indicators for participation (RQ1)

The extent of participation is a simple but essential aspect of collaboration. For collaboration to occur at all, the student must participate in the project. In Tab. 2 basic indicators are described.

Equal participation. In a collaborative software project, individual project members may do most of the work. In an effective collaborative group, all members should participate to a similar degree without monopolizing behavior [5]. Equal participation can be considered by the absolute deviation from the mean. In this way, the scope of individual contributions within a group can be quantified and compared using the indicators shown in Tab. 3

Extent of roles. Related to participation, another factor may regard the variety of roles taken on by the members of the group. A good group should be one in which roles are played flexibly with participants rotating their roles during an iteration. This seems to be indicative of the attention paid to the whole group's process of planning tasks, developing code, and reviewing the quality. In small groups, the extent to which different roles are performed should be independent of assigned roles in agile projects (e.g. product owner) or traditional roles like team leader or quality engineer. The indicators described in Tab. 4 summarize the extent of role fulfilment.

Rhythm. The rhythm of interactions is a metric that allows conclusions to be drawn about the synchronicity of activities. Regular and constant participation can be considered as an indicator of the individual's ability to deal primarily with the needs of the group rather than with personal problems, which also avoids the risk of distraction and a decline in cognitive tension. Considered for this purpose are the indicators in Tab. 5.

3.3 Indicators for cohesion (RQ2)

Table 1: Indicators for software code maintainability

Programming languages (S1): Number of programming languages including style sheet languages in use. In web development, for instance, this indicator can help to identify full-stack developers compared to the one that stick to a single programming language.
Code smells (S2): **(author?)** [13] introduces the metaphor "code smells" to describe the patterns in the code that indicate the need for code refactoring can be applied. The external behavior of the code remains the same when refactoring, while the internal structure improves, i.e., it appears tidier, more traceable, and easier to maintain. This indicator helps teachers and students identify code that needs refactoring. [11, 13]
Cognitive complexity (S3): Cognitive complexity is a measure of the understandability of a given piece of code, the complexity of which is determined by the number and order of control structures [6]. The cognitive complexity can help students develop less nested code. [6]
Security hotspots (S4): The most important security hotspots are authentication, storage, cryptography, logic errors, synchronization and timing, and validation. The security hotspots indicator trains students in recognizing sensitive areas in which security is more important than in other areas. [27]
Vulnerabilities (S5): The number of vulnerabilities that refers to problems in the source code identified from poor coding patterns [13] that lead to bugs, security vulnerabilities, performance problems, design flaws, and other difficulties [29]. In the case of a vulnerability, an issue has been identified that affects the security of the application and must be resolved immediately. For this reason, it is important that students are made aware of vulnerabilities during their training. [13, 29]
Bugs (S6): Number of errors due to a specification that was not adhered to or incorrectly implemented (e.g. typing of return values). To writre productive code, it is important for student to write error free code.
Duplicated lines (S7): The absolute number of physical lines (not just lines of code) of source code that are involved in at least one additional location. Duplicated lines can cause errors to creep into the source code, as changes in the code should always be carried out on all duplicated lines. [7]

Table 2: Basic indicators of participation

Commit Count (P1): One or many changes in the code or file structure can be subitted to the version control system as a revision of the previous state. The level of participation in terms of counting commits is a simple but essential aspect of collaboration. For collaboration to occur at all, the student must participate in the project. [30]
Opened Merge Request Count (P2): In GitLab Merge Request are required to integrate individual contributions into the main code base which is referred as master branch. An opened Merge Request indicates indivual effort to to contribute an individual solved part to the code base. Learners who have opened Merge Requests indicate a good command the collaborative use of git. [30]
Branch Count (P3): A branche indicates separate development line focussing on a particular feature or issue to be implemented. Processing issues and feature request in individual branches indicate a professional use of git in order to separate changes and to avoid merge conflicts. [30]
Comment Count (P4): Issues as well as Merge Requests can be commented and discussed in the team. In remote collaboration comments indicate an effective way to communicate with the team and to document relevant information. [30]
Issue Count (P5): Using a divide and conquere strategy to break-up big tasks into smaller manageable pices which are referred as Issues in GitLab. Defining sub-tasks in terms of Issues enables task delegation and joint development. The ability to define Issues is a fundament skill in IT project management. [30]

Caring for one another among team members is critical to strengthening mutual trust and a sense of belonging, as well as the perception of positive interdependence among members of a group. As shown in Tab. 6, mutual help in programming can be expressed in continuing or improving the work of others.

4. METHODS

Building on the indicators defined in the last section, we describe the technical processing steps for determining the indicators, present a dashboard for monitoring groups, and validate the indicators using data from four student groups.

4.1 Datasets

For the validation, we collected pseudomized data from 11 mixed software development teams that collaborated on a complex task over 7 iterations à 2 weeks as part of the course [COURSE-X] in 2015 to 2022 (cf. Tab. 7). The agile teams worked on different tasks, which were however comparable in terms of effort and demand. The teams consisted of 5 to 6 students from Bachelor's and Master's programs in Computer Science. In total, data from 65 students, 20 of them female, were considered. The age at the time of the course ranges from 24 to 48 (M=33.6, SD=5.7). The regular GitLab Community Edition was used for collaboration. Students consented to the use of GitLab and were aware of the data collected by the system. Each group worked in at least

Table 3: Indicators of equal participation

Equal Commits (P6): Equal participation in terms of the number of created commits can be considered bythe absolute deviation from the mean number of commits in a team. In a collaborative software project, individual project members may do most of the work. In an effective collaborative group, all members should participate to a similar degree without monopolizing behavior. In consequence every team member will get a chance to learn, to have success, and to make mistakes.
Equal Merges (P7): Equal distribution of the number of Merge Requests amoung the team members. Every learner should learn how handle problems that typically arise with Merge Requests.
Equal Issues (P8): Equal distribution of the number of created Issues amoung the team members. All team members should participating in project management by defining sub-tasks such as the Issues in GitLab.
Equal Comments (P9): Equal distribution of the number of comment compared to the group average. Communication should be considered as a reciprocal endavor. All team members should be involved and therefore participate in discussions and the decision making processes.

Table 4: Extent of roles

Active Reviewer (P10): An Active Reviewer is someone who has left a comment on a merge request. This indicator is the ratio of active reviewers to students who have created a merge request. A good group should be one in which roles are played flexibly with participants rotating their roles during an iteration. This seems to be indicative of the attention paid to the whole group's process of planning tasks, developing code, and reviewing the quality. In small groups, the extent to which different roles areperformed should be independent of assigned roles in agile projects (e.g. product owner)or traditional roles like team leader or quality engineer.
Active Developer (P11): The share of team members who created merge requests. In teach software engineering each learner should participate as developer in order to improve and practice its skills.
Active planner (P12): The relative number of team mates created or modified an issue. Planning and decission making should be considered as a joint processes requiring communication.

one GitLab project using the provided issue tracker, version control, wiki, etc. Apart from the data collected and stored in GitLab (including source code), no other data was captured.

4.2 Measurements

Our first attempt to validate the model described in section 3 aims at testing the appropriateness, feasibility, and expressiveness of the indicators.

The indicators can be considered appropriate because they

Table 5: Rhythm of participation

Coding Days (P13): Thr the number of days a developer has contributed code to the project. The rhythm of interactions is a metric that allows conclusions to be drawn about the synchronicity of activities. Regular and constant participation can be considered as an indicator of the individual's ability to deal primarily with the needs of the group rather than with other issues (e.g. other courses), which also avoids the risk of distraction and a decline in cognitive tension.
Review Days (P14): The number of days a team member has reviewed code. Reviews are important in order to ensure the quality of the individual code contributions.
Testing Days (P15): The number of days a team member has tested code. Running test implies that instruments like unit tests have been implemented.
Planning Days (P16): The number of days a team member has been working on project planning in terms of defining issues. Depending on the current role planing can take up a consierable amount of time.

were piloted against the data of the 11 teams in Tab. 7 and discussed with experienced teachers. In addition, the indicators have been adopted from existing tools and research. In terms of feasibility, it must be ensured that all indicators in the model can be calculated using data from real courses. The tests have shown that the calculation is feasible for the datasets at hand. To validate the expressiveness of the indicators as a formative feedback instrument in the supervision of student teams, the indicators should discriminate students to estimate differences among group members. This can be achieved by calculating the standard deviation (SD) among group members and the discrimination index (DI, see [10]) across all indicators. A SD greater zero indicates heterogeneous values for an indicator within a group. If, on the other hand, the values within a group would be very similar, the indicator would have little significance for a teacher.

4.3 Data processing

The presented analytics environment[1] is responsible for storing the transformed raw GitLab data, their analysis, and presentation to the teachers. The component consists of four modules: First, the Extract, Load, Transform (ELT) [17] module performs the daily job of extracting the data from the GitLab projects, loading it to the Data Lake module, and pre-processing it for the analysis. Using SonarQube [29] a code analysis is performed with the GitLab data, before the described indicators are computed. The Data Lake, as the second module, stores the raw data and their transformed version in a relational database schema. For data Analysis the third module performs additional analytics, including data validation as described in section 5.1. Finally, the fourth module consists of a Analytics Dashboard (AD) used by teachers. It is based on the R and Shiny[2] package.

[1]See on GitHub [LINK] (last accessed: 2022-02-27).
[2]See https://shiny.rstudio.com/ (last accessed: 2022-02-27).

Table 6: Indicators for group cohesion

(C1) Commit-Comment-Ratio: Describes the ratio of comments in commits to the number of commits created. In a collaborative project, the proportion of own commits and the number of comments should be balanced to ensure that the student both contributes their own code to the project and is willing to look at other students' code.
Reaction Time (C2): Reaction time is the time it takes a reviewer to respond to a comment directed to them. The other team members should respond to a comment on time so that the questioner does not lose time. The lower the Reaction time the quicker a team is able to process review and integrate new code.
Responsiveness (C3): The response time is the average time taken to respond to a reviewer's comment with either another comment or a code revision. It shows the time between the last comment of the reviewer and the response of the creator of a merge request.
Follow on Commits (C4): The number of code revisions added to a pull request after it was opened for review. Knowing the number of follow-up commits added to an open pull request will give you insight into the strength of your code review process. If you see a trend where many follow-up commits are added, further planning and testing may be required. [22]
Receptiveness (C5): is the frequency with which the creator of the merge request takes comments as a reason to change the code. This in an indicator on how much critique someone accepts in order to improve the code quality. [22]
Time to first Comment (C6): Specifies the time between opening a pull request and the first reviewer commenting. The better a team integrates code, the better it can work together. [22]
Time to resolve (C7): The time it takes to close a pull request. New code needs to be tested an integrated quickly. [22]
Helping others (C8): Describes the percentage of commits that a developer has used to modify the code of his team members (cf. [8].
Reactivity to proposals (C9): Suggestions for new code contributions, which are called merge or pull requests, are essential for progress in a project. Therefore, suggestions from a team member must be immediately considered by the team.

For the AD the data from the Analytics and the Datalake is processed to a web-based interactive visualization. Fig. 1 shows the resulting dashboard for an exemplary group of 6 students that have been collaborated over 3 iterations. The indicators' values have been normalized for the sake of comparison. The data processing for the 11 groups took 12 hours on a Intel Core i5-3210M CPU 2.50 GHz with 16 GB RAM. The Sonarcube scanner took up most of the time.

5. RESULTS

5.1 Validation of indicators

Our first attempt to validate the model described in section 3 aims at testing the appropriateness, feasibility, and expressiveness of the indicators. The number of review and planning days appeared to be often the same for all group members and across all iterations. Certain indicators showed to be less expressive in the last iteration (e.g. C1, C2, C5, and C6) when the groups were mainly focused on bug fixing (e.g. S6). The DI is the correlation of a particular indicator to the total score of all other indicators of a student. Values above .3 are considered good, between .2 and .3 acceptable, between .1 and .2 marginal, and below .1 poor. Indicators that are not sufficiently different from others, i.e., have poor DI, should be removed from the model. Within groups and iterations, no indicators had a poor DI.

Table 7: Overview of the datasets used for validating the model

Group	Semester	N	Reposit.	Issues	Commits
A	2015/16	6	1	350	1,195
B	2016/17	9	1	180	1,129
C	2017/18	5	1	249	878
D	2018/19	3	1	102	233
E	2019/20	7	1	191	1,224
F	2020/21	6	1	192	2,962
G	2020/21	6	1	145	2,419
H	2020/21	5	3	128	1,441
I	2021/22	5	1	260	1,050
J	2021/22	6	8	201	387
K	2021/22	6	3	144	330
Total	7	64	22	2,142	13,248

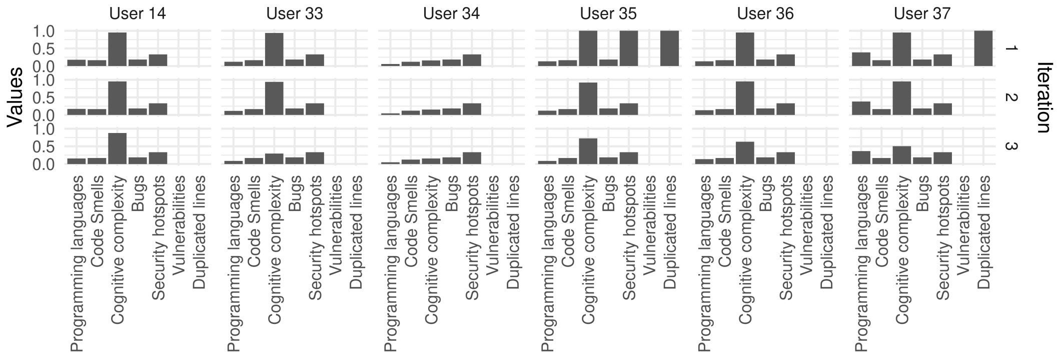
5.2 Teacher dashboard

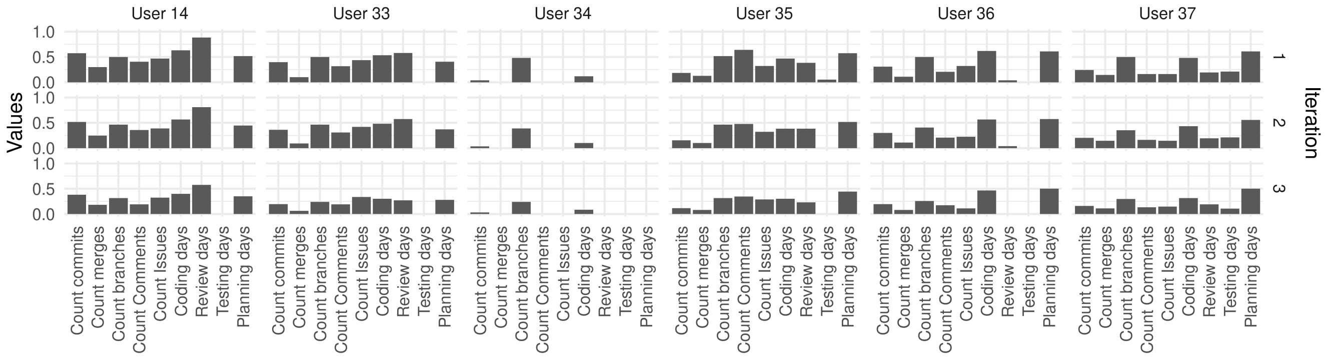
The dashboard as shown in Fig. 1 provides small multiples for comparison of group members with regard to iterations and indicators. For instance, user 33 was commendably able to reduce the cognitive complexity of his code over the three iterations. User 34's comparatively low participation (e.g., P1, P2, P4) should prompt the teacher to ensure learning success. Regarding group cohesion, the teacher could discuss with the team how to achieve mutual help during the programming task, since indicators C1, C2, C3, and C8 have low values.

6. DISCUSSION

In this section, we briefly present and discuss the findings, limitations, and shortcomings of our current approach. We will first address the measured parameters, then the processing, and finally the use of analytics.

For reasons of space, we will limit ourselves to a brief listing regarding the measured parameters. (1) Since we used GitLab as the main environment for collaboration, the team lacked an integrated communication option. Therefore, they communicated outside of GitLab so we could not track it. (2) Commits (e.g. P1, P6, and C4) often contained only small and rarely major changes. (3) In pair programming, usually only the driver made changes, while the observer or navigator did not show up in the commit log (e.g. C8). (4) There was a natural fluctuation in team performance over the course of a semester. (5) The previous point was also influenced by external factors, e.g. holidays, vacations, sick leave. (6) Other study-related tasks or exams and associated workloads were not considered. (7) It was assumed that all students have the same preposition (e.g., skills, experiences,





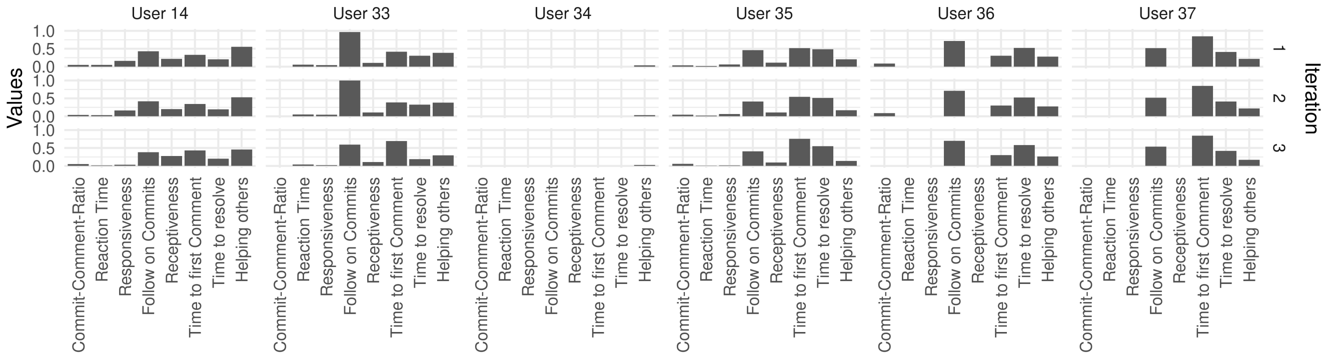


Figure 1: Example output for the first three iterations of group C including the dimensions code quality (top), participation (middle), and cohesion (bottom)

personality [21], health), i.e., related diversity aspects must be taken into account by the teacher. (8) The measured quantities were considered comparable atomic units, i.e. all commits were considered equal even if they differed in terms of number of changes, level of difficulty, and their relevance to the final program (e.g. P1, P6, and C4). (9) Students may have used code snippets or libraries from others (e.g., from the web) without acknowledging or referencing the sources. (10) Since students had to perform all roles at least once, role changes occurred after one iteration, which then led to different behaviours that were not yet considered in the validation of our approach. (11) Compared to many related works (e.g. [14, 32]) we did not classify groups or individuals to provide a flexible tool for different learning and teaching scenarios. Consequently, the teacher has to decide which indicators are relevant for judging groups or learners at a given time.

In terms of the data processing, SonarQube (version 8.9 LTS) covers 27 languages, including CSS, JavaScript, and TypeScript as well as the corresponding analysis rules. Although this is a very good basis, false positives can still occur (e.g. S2, S4, S5, and S6). In this case, the rules used must be understood and adapted if necessary (i.e. by the teachers). Also, the coding conventions defined by the teams (e.g. in ESLint, JSLint) were not considered. For the planning activities, we did not consider the structure of the Kanban board (columns, movement of issues) and the degree of planning (e.g. defining an issue, assigning the issue, estimating the workload, setting a due date) as it could be derived from the data. Besides, solutions are still being sought to avoid subjective assessments due to teachers' implicit bias against females and other underrepresented minorities in teams (e.g. [33]) in combination with teacher dashboards.

7. CONCLUSION AND OUTLOOK

In this paper, we posed 4 research questions, identified, defined, and validated a total of 32 indicators for collaborative learning in software development teams. The resulting model describes collaborative interactions in programming teams considering 16 group participation indicators (RQ1), 9 indicators for group cohesion (RQ2) and 7 indicators for code maintainability (RQ3). In addition, we presented a data processing pipeline for extracting, calculating, and visualizing these indicators on a teacher dashboard (RQ4). This approach enables teachers to keep track of complex group activities and individual contributions and then provide targeted formative feedback to the groups.

Next semester, we plan to use the tool as a monitoring instrument in a programming course. We hope that the tool will help teachers to provide better guidance and support regarding programming performance, participation in learning as well as group climate. So far, we focused on an iteration-based analysis. We plan to update the model and analysis to support three additional features: analysis of all previous iterations, relative comparisons between dedicated iterations either per person or per team, and individual trend analysis as well as enabling the teacher to dive into representative code segments.

References

[1] K. Agrawal, S. Amreen, and A. Mockus. Commit Quality in Five High Performance Computing Projects. In *Proceedings of the 2015 International Workshop on Software Engineering for High Performance Computing in Science*, SE4HPCS '15, pages 24–29. IEEE Press, 2015.

[2] F. Ahmed, L. F. Capretz, S. Bouktif, and P. Campbell. Soft skills and software development: A reflection from software industry. *International Journal of Information Processing and Management*, 4(3):171–191, 2013.

[3] L. Ardito, R. Coppola, L. Barbato, and D. Verga. *A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review.* Aug. 2020. Publication Title: Scientific Programming Type: Review Article.

[4] K. Buffardi. Assessing Individual Contributions to Software Engineering Projects with Git Logs and User Stories. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, pages 650–656, New York, NY, USA, 2020. ACMy.

[5] A. Calvani, A. Fini, M. Molino, and M. Ranieri. Visualizing and monitoring effective interactions in online collaborative groups. *British Journal of Educational Technology*, 41(2):213–226, 2010.

[6] G. A. Campbell. Cognitive complexity: an overview and evaluation. In *Proceedings of the 2018 International Conference on Technical Debt*, TechDebt '18, pages 57–58, New York, NY, USA, May 2018. ACMy.

[7] G. A. Campbell and P. P. Papapetrou. *SonarQube in Action.* Manning Publications Co., USA, 1st edition, 2013.

[8] P. H. Center. What is Impact?, 2021.

[9] D. Čubranić and M. A. D. Storey. Collaboration Support for Novice Team Programming. In *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*, GROUP '05, pages 136–139, New York, NY, USA, 2005. ACMy.

[10] A. F. De Champlain. A primer on classical test theory and item response theory for assessments in medical education. *Medical education*, 44(1):109–117, jan 2010.

[11] E. v. Emden and L. Moonen. Java quality assurance by detecting code smells. In *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, pages 97–106, Nov. 2002.

[12] S. Eraslan, J. C. C. Rios, K. Kopec-Harding, S. M. Embury, C. Jay, C. Page, and R. Haines. Errors and Poor Practices of Software Engineering Students in Using Git. In *Proceedings of the 4th Conference on Computing Education Practice 2020*, CEP 2020, New York, NY, USA, 2020. ACMy.

[13] M. Fowler. *Refactoring: Improving the Design of Existing Code [Book].* 1998.

[14] N. Gitinabard, R. Okoilu, Y. Xu, S. Heckman, T. Barnes, and C. Lynch. Student Teamwork on Programming Projects: What can GitHub logs show us?, 2020.

[15] R. Glassey. Adopting Git/Github within Teaching: A Survey of Tool Support. In *Proceedings of the ACM Conference on Global Computing Education*, CompEd '19, pages 143–149, New York, NY, USA, 2019. ACMy.

[16] C. Gote, I. Scholtes, and F. Schweitzer. Git2net: Mining Time-Stamped Co-Editing Networks from Large Git Repositories. In *Proceedings of the 16th International Conference on Mining Software Repositories*, MSR '19, pages 433–444. IEEE Press, 2019.

[17] V. Gour, S. Sarangdevot, G. S. Tanwar, and A. Sharma. Improve performance of extract, transform and load (etl) in data warehouse. *International Journal on Computer Science and Engineering*, 2(3):786–789, 2010.

[18] M. Hoegl and H. G. Gemuenden. Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence. *Organization Science*, 12(4):435–449, Aug. 2001.

[19] M. Martinez and M. Monperrus. Coming: A Tool for Mining Change Pattern Instances from Git Commits. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, ICSE '19, pages 79–82. IEEE Press, 2019.

[20] J. Menezes, C. Gusmão, and H. Moura. Risk factors in software development projects: a systematic literature review. *Software Quality Journal*, 27(3):1149–1174, 2019.

[21] I. Nunes, C. Treude, and F. Calefato. The Impact of Dynamics of Collaborative Software Engineering on Introverts: A Study Protocol. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, pages 619–622, New York, NY, USA, 2020. ACMy.

[22] Pluralsight. *Flow metrics Pluralsight Help Center*. 2021.

[23] J. C. C. Rios, K. Kopec-Harding, S. Eraslan, C. Page, R. Haines, C. Jay, and S. M. Embury. A Methodology for Using GitLab for Software Engineering Learning Analytics. In *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '19, pages 3–6. IEEE Press, 2019.

[24] C. Rosen, B. Grawi, and E. Shihab. Commit Guru: Analytics and Risk Prediction of Software Commits. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 966–969, New York, NY, USA, 2015. ACMy.

[25] M. Rosli, E. Tempero, and A. Luxton-reilly. A Systematic Mapping Study on Data Quality in Software Engineering Data Sets. *Journal of Universal Computer Science*, 25(1):16–41, 2019.

[26] J. J. Sandee and E. Aivaloglou. GitCanary: A Tool for Analyzing Student Contributions in Group Programming Assignments. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, Koli Calling '20, New York, NY, USA, 2020. ACMy.

[27] A. Sensaoui, O.-E.-K. Aktouf, D. Hely, and S. Di Vito. An In-depth Study of MPU-Based Isolation Techniques. *Journal of Hardware and Systems Security*, 3(4):365–381, Dec. 2019.

[28] L. Silva, A. J. Mendes, and A. Gomes. Computer-supported collaborative learning in programming education: A systematic literature review. *IEEE Global Engineering Education Conference, EDUCON*, 2020-April(May):1086–1095, 2020.

[29] SonarQube. Code Quality and Code Security, 2021.

[30] D. A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman. Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering*, 24(3):1369–1417, June 2019.

[31] N. Tsantalis, M. Mansouri, L. M. Eshkevari, D. Mazinanian, and D. Dig. Accurate and Efficient Refactoring Detection in Commit History. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 483–494, New York, NY, USA, 2018. ACMy.

[32] M. Tushev, G. Williams, and A. Mahmoud. Using GitHub in large software engineering classes. An exploratory case study. *Computer Science Education*, 30(2):155–186, apr 2020.

[33] L. van den Bergh, E. Denessen, L. Hornstra, M. Voeten, and R. W. Holland. The Implicit Prejudiced Attitudes of Teachers: Relations to Teacher Expectations and the Ethnic Achievement Gap. *American Educational Research Journal*, 47(2):497–527, jun 2010.

[34] M. Wessel. Leveraging Software Bots to Enhance Developers' Collaboration in Online Programming Communities. In *Conference Companion Publication of the 2020 on Computer Supported Cooperative Work and Social Computing*, CSCW '20 Companion, pages 183–188, New York, NY, USA, 2020. ACMy.