

Indicators of group learning in collaborative software development teams




Benjamin Weiher¹, Niels Seidel¹ , Marc Burchart¹ , Dirk Veiel² 


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 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 7 indicators for code quality, 16 group participation indicators, and 9 indicators for group cohesion. 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; 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 [Ah13, RTLr19]. 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) [SMG20]. 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. [G19]). These systems generate data that can be used not only for risk analysis of projects (e.g. [MGM19]) but also to support students working together on a software project.

¹ FernUniversität in Hagen, Faculty of Mathematics and Computer Science, Universitätsstraße 1, 58084 Hagen, benjamin.weiher@studium.fernuni-hagen.de, niels.seidel@fernuni-hagen.de, marc.burchart@fernuni-hagen.de,  Niels Seidel <https://orcid.org/0000-0003-1209-5038>,  Marc Burchart <https://orcid.org/0000-0001-5668-7137>, 

² FernUniversität in Hagen, Research Cluster D²L², Universitätsstraße 27, 58084 Hagen, dirk.veiel@fernuni-hagen.de,  Dirk Veiel <https://orcid.org/0000-0003-0228-103X>

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 learning analytics. Our contribution to CSCL consists of a model describing collaborative interactions in programming teams considering coding, group participation, and group cohesion. Regarding learning analytics, 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 [Ri19, SA20, Bu20]. However, these only consider individual contributions, not the group collaboration. [Gi20] 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 [TWM20] 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, [TWM20] 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 [GSS19], commit quality [AAM15], refactoring detection [Ts18], change patterns [MM19], and risky commit prediction [RGS15], 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 [Er20]. Personal assistance systems support collaboration only on a low-level [ČS05] or even hinder it [We20], 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 [Ca10] on effective group models and the considerations of [HG01] 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. [Ar20]) are suitable for the formative analysis and assessment of learning achievements in programming. We consider the indicators S1–S7 to be relevant in order to answer RQ 3: (S1) *Programming languages*: 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. (S2) *Code smells*: [Fo98] introduces the metaphor “code smells” to describe the patterns in the code that indicate the need for code refactoring. 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 [EM02]. (S3) *Cognitive complexity*: 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 [Ca18]. (S4) *Security hotspots*: The security hotspot describes the sensitive areas where security is more important than in other areas. The most important security hotspots are authentication, storage, cryptography, logic errors, synchronization and timing, and validation [Se19]. (S5) *Vulnerabilities*: The number of vulnerabilities that refers to problems in the source code identified from poor coding patterns [Fo98] that lead to bugs, security vulnerabilities, performance problems, design flaws, and other difficulties [So21]. (S6) *Bugs*: Number of errors due to a specification that was not adhered to or incorrectly implemented (e.g. typing of return values). (S7) *Duplicated lines*: The absolute number of physical lines (not just lines of code) of source code that are involved in at least one additional location [CP13].

3.2 Indicators for participation (RQ1)

Extent of participation 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. Following [Ta19], *Commit Count* (P1), *Opened Merge Request Count* (P2), *Branch Count* (P3), *Comment Count* (P4), and *Issue Count* (P5) has to be recorded.

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 [Ca10]. 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 *Equal Commits* (P6), *Equal Merges* (P7), *Equal Issues* (P8), and *Equal Comments* (P9).

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. In this sense, we define an *Active Reviewer* (P10) as 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. The share of *Active Developer* (P11) in a team considers those who created merge requests. The relative number of *Active planner* (P12) created or modified an issue.

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 *Coding Days* (P13), *Review Days* (P14), *Testing Days* (P15), and *Planning Days* (P16). Coding Days, for instance, counts the number of days a developer has contributed code to the project. Similarly, this applies to other important team tasks such as planning, reviewing each other's work, and testing.

3.3 Indicators for cohesion (RQ2)

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. (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. (C2) *Reaction Time*: Response 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. (C3) *Responsiveness*: 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. (C4) *Follow on Commits*: 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 [PI21]. The *Receptiveness* (C6) is the frequency with which the creator of the merge request takes comments as a reason to change the code (cf. [PI21]). In addition to that, the *Time to first Comment* (C6) specifies the time between opening a pull request and the first reviewer commenting (cf. [PI21]) and the *Time to resolve* (C7) is the time it takes to close a pull request (cf. [PI21]).

Mutual help in programming can be expressed in continuing or improving the work of others. Therefore, the indicator *Helping others* (C8) describes the percentage of commits that a developer has used to modify the code of his team members (cf. [PI21]). *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.

4 Realisation and validation

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 Analytics environment

The analytics environment 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) [Go10] 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 [So21] 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 4.2. Finally, the fourth module consists of a Analytics Dashboard (AD) used by teachers. It is based on the R and Shiny³ package. 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 visualization as small multiples enables a comparison between students on each iteration and for all indicators. For

³ See <https://shiny.rstudio.com/> (last accessed: 2021-06-27).

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.

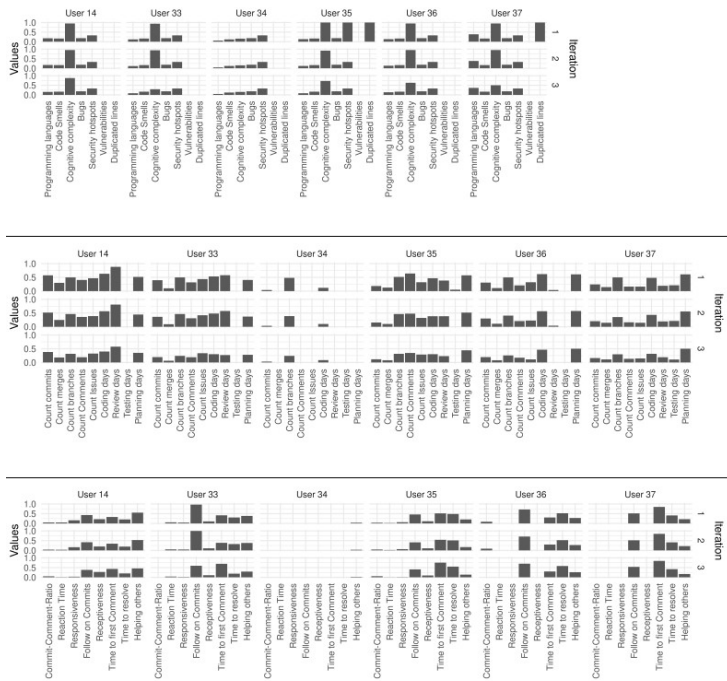


Fig. 1: Example output for the first three iterations of group C including the dimensions code quality (top), participation (middle), and cohesion (bottom)

4.2 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. For the validation, we used data from four mixed software development teams that worked on a complex task over 7 iterations à 2 weeks as part of the course “Fachpraktikum CSCW” (cf. Tab. 1). The agile teams worked on different tasks, which were however comparable in terms of effort and demand, in regular dialog with a role-played customer. The teams consisted of 5 to 6 students from Bachelor’s and Master’s programs in Computer Science. GitLab was used for collaboration. Each group worked in at least one GitLab project using provided issue tracker, version control, etc.

Group	Semester	Participants	Repositories	Issues	Commits
A	WS 2019/20	5	1	191	4402
B	WS 2020/21	5	1	192	2962
C	WS 2020/21	6	1	145	2419
D	WS 2020/21	5	3	128	1441

Tab. 1: Overview of the datasets used for validating the model

The indicators can be considered appropriate because they were piloted against the data of the teams in Tab. 1 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 [De10]) across all indicators. The SD indicates heterogeneous values of an indicator within a group. If, on the other hand, the values within a group were always the same, the indicator would have little significance for a teacher. 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.

4.3 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, personality [NTC20], 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. [Gi20, TWM20]) 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. [Be10]) in combination with teacher dashboards.

5 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.

Acknowledgments

This research was supported by the Research Cluster “Digitalization, Diversity and Lifelong Learning – Consequences for Higher Education” (D²L²) of the FernUniversität in Hagen, Germany.

Bibliography

- [AAM15] Agrawal, Kamil; Amreen, Sadika; Mockus, Audris: 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. IEEE Press, pp. 24–29, 2015.
- [Ah13] Ahmed, Faheem; Capretz, Luiz Fernando; Bouktif, Salah; Campbell, Piers: Soft skills and software development: A reflection from software industry. *International Journal of Information Processing and Management*, 4(3):171–191, 2013.
- [Ar20] Ardito, Luca; Coppola, Riccardo; Barbato, Luca; Verga, Diego: A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review. August 2020. Publication Title: Scientific Programming Type: Review Article.
- [Be10] van den Bergh, Linda; Denessen, Eddie; Hornstra, Lisette; Voeten, Marinus; Holland, Rob W: 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.
- [Bu20] Buffardi, Kevin: 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. ACM, New York, NY, USA, pp. 650–656, 2020.
- [Ca10] Calvani, Antonio; Fini, Antonio; Molino, Marcello; Ranieri, Maria: Visualizing and monitoring effective interactions in online collaborative groups. *British Journal of Educational Technology*, 41(2):213–226, March 2010.

-
- [Ca18] Campbell, G. Ann: Cognitive complexity: an overview and evaluation. In: Proceedings of the 2018 International Conference on Technical Debt. TechDebt '18, ACM, New York, NY, USA, pp. 57–58, May 2018.
- [CP13] Campbell, G. Ann; Papapetrou, Patroklos P.: SonarQube in Action. Manning Publications Co., USA, 1st edition, 2013.
- [ČS05] Čubranić, Davor; Storey, Margaret Anne D: Collaboration Support for Novice Team Programming. In: Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work. GROUP '05, ACM, New York, NY, USA, pp. 136–139, 2005.
- [De10] De Champlain, André F: A primer on classical test theory and item response theory for assessments in medical education. *Medical education*, 44(1):109–117, jan 2010.
- [EM02] Emden, E. van; Moonen, L.: Java quality assurance by detecting code smells. In: Ninth Working Conference on Reverse Engineering, 2002. Proceedings. pp. 97–106, November 2002.
- [Er20] Eraslan, Sukru; Rios, Julio César Cortés; Kopec-Harding, Kamilla; Embury, Suzanne M; Jay, Caroline; Page, Christopher; Haines, Robert: Errors and Poor Practices of Software Engineering Students in Using Git. In: Proceedings of the 4th Conference on Computing Education Practice 2020. CEP 2020, ACM, New York, NY, USA, 2020.
- [Fo98] Fowler, Martin: Refactoring: Improving the Design of Existing Code [Book]. 1998.
- [Gi20] Gitinabard, Niki; Okoilu, Ruth; Xu, Yiqao; Heckman, Sarah; Barnes, Tiffany; Lynch, Collin: . Student Teamwork on Programming Projects: What can GitHub logs show us?, 2020.
- [Gl19] Glassey, Richard: Adopting Git/Github within Teaching: A Survey of Tool Support. In: Proceedings of the ACM Conference on Global Computing Education. CompEd '19, ACM, New York, NY, USA, pp. 143–149, 2019.
- [Go10] Gour, Vishal; Sarangdevot, SS; Tanwar, Govind Singh; Sharma, Anand: Improve performance of extract, transform and load (ETL) in data warehouse. *International Journal on Computer Science and Engineering*, 2(3):786–789, 2010.
- [GSS19] Gote, Christoph; Scholtes, Ingo; Schweitzer, Frank: Git2net: Mining Time-Stamped Co-Editing Networks from Large Git Repositories. In: Proceedings of the 16th International Conference on Mining Software Repositories. MSR '19. IEEE Press, pp. 433–444, 2019.
- [HG01] Hoegl, Martin; Gemuenden, Hans Georg: Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence. *Organization Science*, 12(4):435–449, August 2001.
- [MGM19] Menezes, Júlio; Gusmão, Cristine; Moura, Hermano: Risk factors in software development projects: a systematic literature review. *Software Quality Journal*, 27(3):1149–1174, 2019.
- [MM19] Martinez, Matias; Monperrus, Martin: 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. IEEE Press, pp. 79–82, 2019.

- [NTC20] Nunes, Ingrid; Treude, Christoph; Calefato, Fabio: 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, ACM, New York, NY, USA, pp. 619–622, 2020.
- [Pl21] Pluralsight: Flow metrics Pluralsight Help Center. 2021.
- [RGS15] Rosen, Christoffer; Grawi, Ben; Shihab, Emad: 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, ACM, New York, NY, USA, pp. 966–969, 2015.
- [Ri19] Rios, Julio César Cortés; Kopec-Harding, Kamilla; Eraslan, Sukru; Page, Christopher; Haines, Robert; Jay, Caroline; Embury, Suzanne M: 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. IEEE Press, pp. 3–6, 2019.
- [RTLr19] Rosli, Marshima; Tempero, Ewan; Luxton-reilly, Andrew: A Systematic Mapping Study on Data Quality in Software Engineering Data Sets. Journal of Universal Computer Science, 25(1):16–41, 2019.
- [SA20] Sandee, Jan Jaap; Aivaloglou, Efthimia: 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, ACM, New York, NY, USA, 2020.
- [Se19] Sensaoui, Abderrahmane; Aktouf, Oum-El-Kheir; Hely, David; Di Vito, Stephane: An In-depth Study of MPU-Based Isolation Techniques. Journal of Hardware and Systems Security, 3(4):365–381, December 2019.
- [SMG20] Silva, Leonardo; Mendes, Antonio Jose; Gomes, Anabela: Computer-supported collaborative learning in programming education: A systematic literature review. IEEE Global Engineering Education Conference, EDUCON, 2020-April(May):1086–1095, 2020.
- [So21] SonarQube: . Code Quality and Code Security, 2021.
- [Ta19] Tamburri, Damian A.; Palomba, Fabio; Serebrenik, Alexander; Zaidman, Andy: Discovering community patterns in open-source: a systematic approach and its evaluation. Empirical Software Engineering, 24(3):1369–1417, June 2019.
- [Ts18] Tsantalis, Nikolaos; Mansouri, Matin; Eshkevari, Laleh M; Mazinianian, Davood; Dig, Danny: Accurate and Efficient Refactoring Detection in Commit History. In: Proceedings of the 40th International Conference on Software Engineering. ICSE '18, ACM, New York, NY, USA, pp. 483–494, 2018.
- [TWM20] Tushev, Miroslav; Williams, Grant; Mahmoud, Anas: Using GitHub in large software engineering classes. An exploratory case study. Computer Science Education, 30(2):155–186, apr 2020.
- [We20] Wessel, Mairieli: 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, ACM, New York, NY, USA, pp. 183–188, 2020.