http://doi.org/10.23925/2179-3565.2024v15i2p152-162

RISUS - Journal on Innovation and Sustainability
volume 15, number 2 - 2024
ISSN: 2179-3565
Editor Científico: Arnoldo José de Hoyos Guevara
Editor Assistente: Vitória Catarina Dib
Avaliação: Melhores práticas editoriais da ANPAD

SOCIAL NETWORK ANALYSIS APPLIED AT UNDERSTANDING COLLABORATION IN SOFTWARE PROJECTS

Análise de redes sociais aplicada ao entendimento da colaboração em projetos de software

Fabio R. de Miranda[1,2], Marcelo A. V. Graglia[1]
[1]Pontifical Catholic University of São Paulo, TIDD, Brazil
[2]Institute of Education and Research, INSPER, São Paulo, Brazil
E-mail: fabiomiranda@insper.edu.br, ra00342850@pucsp.edu.br, mgraglia@pucsp.br

ABSTRACT

In the realm of software development, collaboration is important for worker satisfaction, mitigating turnover risks and better design of final products. The quality of collaboration in teams can indirectly assessed by the interdependence of team members' designs and the extent of shared authorship within a system. Our goal is to apply Social Network Analysis to visualize collaboration on software teams. This study is situated within the framework of software projects undertaken by undergraduate Computer Science students in a 3-week programming sprint. Detailed data from the software development process was gathered through Github and further analyzed using a two-mode social network and later a regular social network. Our results indicate that these techniques help illuminate certain facets of collaboration, such as members that are isolated from the team or collaborators that tend to concentrate too much work.

Keywords: Collaboration; Social network analysis; Software development; Software repositories.

ACEITO EM: 20/05/2024
PUBLICADO EM: 01/08/2024

RISUS - Journal on Innovation and Sustainability
volume 15, número 2 - 2024
ISSN: 2179-3565
Editor Científico: Arnoldo José de Hoyos Guevara
Editor Assistente: Vitória Catarina Dib
Avaliação: Melhores práticas editoriais da ANPAD

ANÁLISE DE REDES SOCIAIS APLICADA AO ENTENDIMENTO DA COLABORAÇÃO EM PROJETOS DE SOFTWARE

Social network analysis applied at understanding collaboration in software projects

Fabio R. de Miranda[1,2], Marcelo A. V. Graglia[1]
[1]TIDD – PUCSP, Pontifical Catholic University of São Paulo, Brazil
[2]Insper Institute of Education and Research, São Paulo, Brazil
E-mail: fabiomiranda@insper.edu.br, ra00342850@pucsp.edu.br, mgraglia@pucsp.br

RESUMO

No âmbito do desenvolvimento de software, a colaboração é importante para a satisfação do trabalhador, mitigando os riscos de rotatividade e melhor design dos produtos finais. A qualidade da colaboração em equipes pode ser avaliada indiretamente pela interdependência dos desenhos dos membros da equipe e pela extensão da autoria compartilhada dentro de um sistema. Nosso objetivo é aplicar a Análise de Redes Sociais para visualizar a colaboração em equipes de software. Este estudo está situado no âmbito de projetos de software realizados por estudantes de graduação em Ciência da Computação em um sprint de programação de 3 semanas. Dados detalhados do processo de desenvolvimento de software foram coletados através do Github e posteriormente analisados usando uma rede social de dois modos e, posteriormente, uma rede social regular. Nossos resultados indicam que essas técnicas ajudam a iluminar certas facetas da colaboração, como membros isolados da equipe ou colaboradores que tendem a concentrar muito trabalho.
Palavras-chave: Colaboração; Análise de redes sociais; Desenvolvimento de software; Repositórios de software.

INTRODUCTION

Collaboration is an important component of teamwork. Research in management suggests that teams that collaborate effectively perform better in highly complex and high-tech projects, as is typically the case with software (Brusoni et al., 2001). Many make the case that every profession must be ready to work in software. Programming will be increasingly a fundamental skill for all professionals, being valuable valuable across a wide range of professions, including finance, marketing, healthcare, and even the arts (World Economic Forum, 2020). In programming team-based projects are becoming the standard. Working together brings numerous advantages; it makes maintaining software easier, encourages knowledge sharing, and helps systems to improve over time. Programmers working in teams often feel more satisfied and experience better well-being (Kropp et al., 2020).

An essential tool for collaborative software projects is a version control system. Such systems serve as repositories for software, recording every developer's contribution. This comprehensive record-keeping is not just for supervision and documentation —it's a valuable resource for analyzing how developers work together. The history of a software evolution monitored through its repository constitutes data that can be represented through a social network.

This work focuses applies social network analysis (SNA) metrics to student developers collaborating on a real software project for an actual client. In this setting, it's important to understand how team members interact and work together. We'll look into the collaboration dynamics within the team to gain insights into how students handle real-world software development projects as a team. Research in management suggests that teams that collaborate effectively perform better in highly complex and high-tech projects, as is typically the case with software.

The objective of this work is applying social network metrics and visualizations to a team project and see if those have correspondences with certain types of contribution patterns between the collaborators.

1 COLLABORATION AND WELL BEING

Professionals who develop software have access to tools that facilitate project task division. Among these, version control systems stand out, especially the Git system. This feature allows more than one developer to work on the project simultaneously. This sharing method introduces different organizational possibilities for collaborative work: at one end, there can be a simple division of tasks where each individual works on a separate part; at the other, a more intense collaboration might occur with shared responsibility and joint authorship.

Qualifying the degree of collaboration on software projects and providing feedback on the maturity level of the collaboration can be seen only as a way to increase productivity. But more than putting programmers in the place of a cog in the machine of the software industry, collaborating goes beyond just enabling productivity and mitigating turnover risks. It can increase satisfaction and well being. Agile software development methodologies, that are collaborative, are associated with higher job satisfaction among developers, as discussed by Kropp et al. (2020). Conversely, attributes of high-quality collaboration such as team learning can only occur when there is psychological safety and absence of conflict (Pinheiro et al., 2023). Agile practices, such as iterative development, frequent communication, and collaboration, create a more engaging and fulfilling work environment. The flexibility and autonomy offered by agile methodologies can contribute to developer satisfaction. Moe et al. (2009) conducted a survey among 34 developers belonging to five teams from three different companies, totaling 149 periodical interviews. That survey found that work teams in software development can lead to increased productivity, innovation, and employee satisfaction.

Kropp et al. (2020) conducted a nationwide survey of software developers in Switzerland found out that being part of collaborative processes are closely related to workers' satisfaction. Another finding is that collaborator's awareness of the structure of collaboration plays a role in satisfaction. Thus, tools that mediate collaboration help boost that awareness. The use of technological tools to mediate collaboration can ease the burden of negotiating internally in teams and defining certain roles and protocols. The work of Andres (2012) identified that the use of such tools increases productivity and team process satisfaction.

The quality of teamwork, which depends on collaboration, was found by Hoegl & Gemuenden (2001) to be associated with team members' work satisfaction and ability to learn in software development. Some agile methodologies such as ASEST (Avila et al., 2021) and ASEST+ (Avila et al., 2022) go besides ceremonies for getting the software done and include mechanisms to incorporating disagreements, construction of consensus and improving the cohesion of the team.

Collaboration is crucial for developing complex software. One reason is that an individual's execution capability cannot handle the complexities of modern software, and beyond a certain complexity, it becomes necessary to work in teams that need to collaborate functionally.

According to programming language pioneer David Parnas, Software Engineering is the development of multi-version programs made by multiple people (Parnas, 2011). This statement underscores that team collaboration is a vital skill for professional software development.

Surveys suggest that experienced software designers are characterized by knowing how to cooperate (Sonnentag, 2000), and they use collaboration as part of their plan to complete a task (Sonnentag, 1998).

Whitehead et al. (2010) emphasize that software project management should establish organizational structures that encourage collaboration, especially vital in a context where the current way of developing software introduces many types of distance within development teams, including spatial, temporal, and socio-cultural. The software quality can also be related to cooperation among development team members (Weimar et al., 2013). Over the past 20 years, the rise of agile software development methodologies has emphasized individual interactions over the uncritical use of processes and tools (Beck et al., 2001).

Merely assigning group tasks to developers or students doesn't usually ensure that collaborative skills are developed or that groups work productively (Bacon et al., 1999).

Katzenbach & Smith (1993) define that a team goes beyond just a group of people who work together. There's an expectation of joint contribution and collaboration, as well as shared accountability for results. It's characteristic of collaborations to deliver projects that benefit from the joint contribution of its members. The Association for Computing Machinery (ACM, 2013) suggests integrating professional software development practices, including collaborative teamwork, into higher education in computing.

2 VERSION CONTROL SYSTEMS AND COLLABORATION

In professional software development and practical projects at the university, tools are used that can maintain a detailed record of how each contributor worked and which part of the software they submitted. This detailed information can be used to measure and encourage collaboration.

While collaboration is a desired skill, it's difficult to precisely measure and quantify. Simple collaboration metrics, such as lines of code (Gousios et al., 2008) are easy to measure and manipulate and can create perverse incentives, falsifying true collaboration.

Contemporary software development projects are highly amenable to ongoing analysis, as they digitally encapsulate every contributor's input through version control systems (VCS). These systems are ubiquitously implemented due to their multiple advantages. One significant benefit is their ability to allow concurrent file editing by multiple programmers without the risk of overwriting each other's contributions. They are designed with conflict resolution mechanisms, prompting manual review in scenarios where developers concurrently modify identical lines of code.

The software project's work history is important to understand the actual collaboration that took place and monitor the process systematically.

Software repository mining (Chaturvedi et al., 2013; Vidoni, 2022) consists of analyzing project history and source code to understand authorship, history, and development evolution. Extracted repository data, using data mining techniques, can lead to understanding and even predictions about the software and its context.

One strategy to achieve collaboration is to measure each individual's contribution and create incentives for collaboration. Software repositories allow for detailed tracking. Extracting quality data from software repositories and using it to understand team dynamics is an active research area in Software Engineering and Computing.

At first glance, it is easier to extract basic metrics, like lines of code or the number of commits. However, the number of code submissions or lines of code are simple quantitative metrics of software development progress and do not necessarily reflect the significance or magnitude of the contributor's input (Jorgensen & Shepperd, 2007). The need to better quantify these contributions has led to more sophisticated metrics, such as function points

(Hira & Boehm, 2016) or cyclomatic complexity (McCabe, 1976). This latter measure analyzes changes in source code by representing the program as a graph, making structural versus incremental changes more apparent. When there is data about the software process, requirements, bugs, and customer wishes, these can also be used as metrics of each developer's individual contribution (Wu et al., 2016). This is an active research area, and a trend is to use artificial intelligence and machine learning to assess a developer's project contribution magnitude, with existing results in this direction (Pillai et al., 2017; Ratner et al., 2017).

Social network analysis is another tool that can be used to analyze the dynamics of teams of collaborators. SNA evaluates the relationships between individuals and entities rather than examining subjects in isolation, and has been used in some context to analyze collaboration. For instance, Riquelme et al. (2019) analyzed interactions in a discussion group in order to build a social network where nodes were participants and the edges between them referred to the strength of the connection. In software development projects, SNA potentially improve understanding of collaboration, and lead to improvements in teamwork, which in turn leading to increased productivity, quality, and knowledge sharing (Costa et al., 2014). In the context of software, social networks have been used in other to study the interactions and visualize communities (Gilbert & Karahalios, 2009; Jorgensen & Shepperd, 2007; Robles et al., 2022).

3 CASE STUDY INVESTIGATING COLLABORATION THROUGH SNA

This work applies social network analysis (SNA) to a software project developed by a team of computer science students, who assumed the roles of collaborators. The analysis aims to identify collaboration patterns and provide insights into the team's work dynamics.

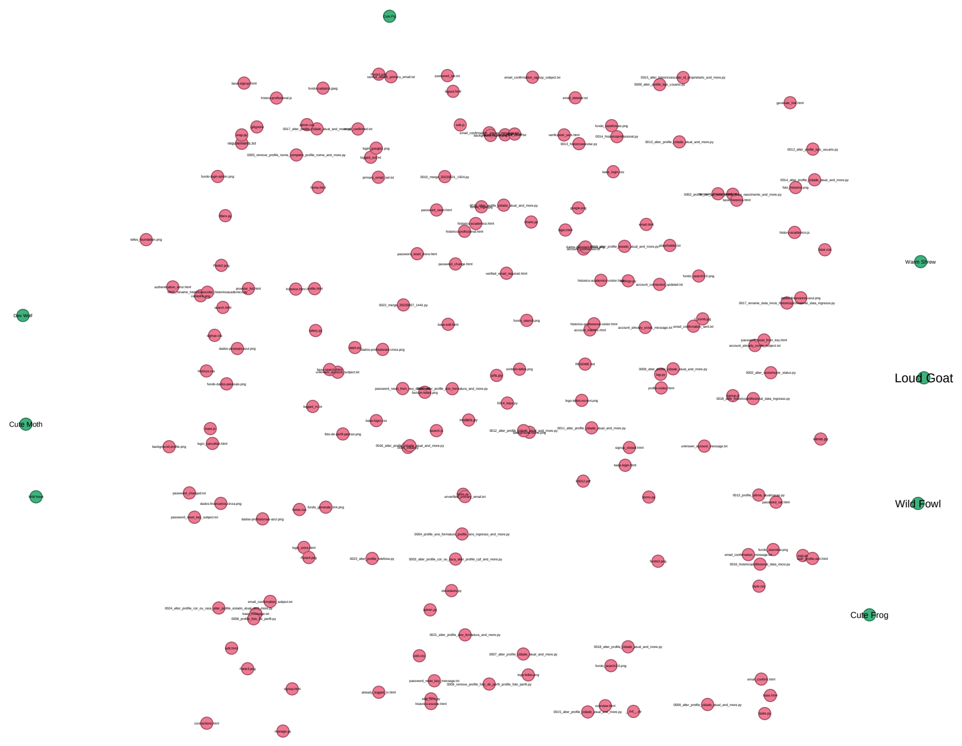
The collaborators, junior computer science students, participated in a 3-week intensive course dedicated to developing a project for an external organization outside the university. Throughout this document, these students are referred to interchangeably as collaborators or programmers, given the nature of the software project.The project's goal was to create a system to track scholarship recipients, ensuring regular updates on their academic progress and status, including grades, internships and expected graduation date. It was expected of the system to come up with measures to improve compliance, such as automatically asking the scholarship recipites for updates through WhatsApp.

The project took the final form of a website backed by a database, programmed in the Python programming language. Being a web project, the files were a mix of Python source code, HTML formatting files and graphic files. The project of the team was comprised of 178 files total. Those files had 862 changes made to them cumulatively over the course of 2.5 weeks.

This took place inside a course in the computer science curriculum, and the students devoted at least 6 hours per day to the project. It took place on a period dedicated to that project and without regular classes occurring at the same time. The project had four faculty members supervising all groups, which were six in total. Though the social network analysis seen here focused on a single group.The group under analyss had seven students/collaborators. Their names have been changed to random animal names in the tables and graphs presented here. Names are consistent over analysis.

Within the scope of this analysis, the students employed Git, a widely-recognized source code version control system, of which the most used service is hosted on GitHub. The programmers were prompted to regularly commit their modifications to the repository, ensuring a comprehensive and retrievable record of the project's evolution. This archival feature not only maintains a historical narrative of the development process but also provides the flexibility to revert changes or restore previous states of the project if required.

Figure 1 - Two-mode network connecting collaborators to project files. Collaborators are green nodes and files are red

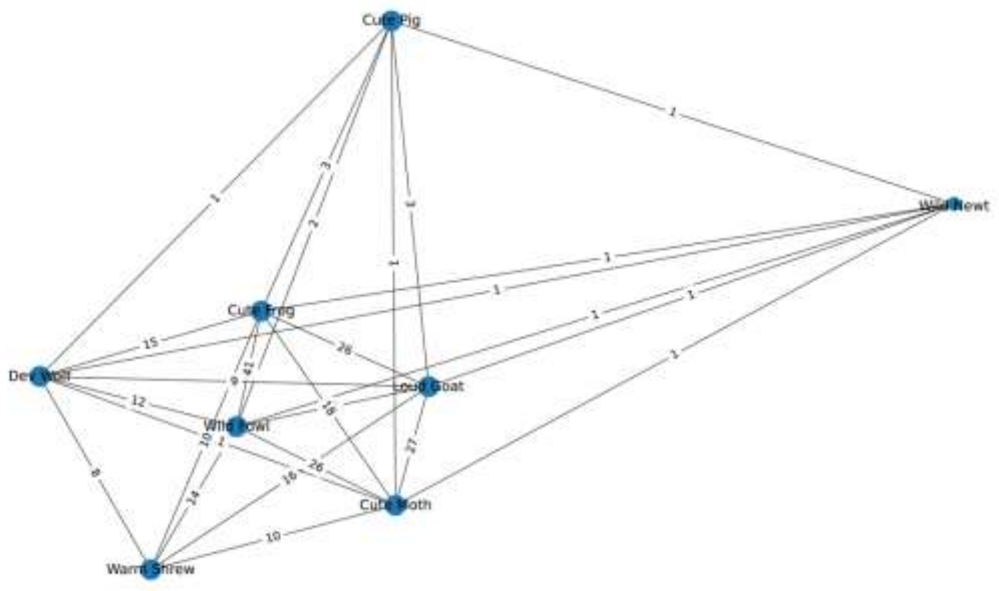


The Git repositories of the project is private because the client preferred it this way. The analysis made in this work was done in Python, using the library Pydriller (Spadini et al., 2018) to extract Github commit data and the library NetworkX to build the social networks and compute SNA metrics. Contribution commit data contained the collaborator id, the time, and the changes made to the file.

In the software project explored in this analysis, the social network does not form direct links between project collaborators. Instead, interactions occur through modifications to project files. Such networks, where one category of participants (in this case, the collaborators) are indirectly linked through another category (here, the files), are identified as two-mode networks (Borgatti & Everett, 1997). These networks often later are converted into a secondary social network that do contain direct connections between entities of the same kind.

In this study, a two-mode network initially maps the interactions of collaborators with files, as can be seen in Figure 1, where collaborators appear as green nodes more along the periphery, and project files are more in the center. The size of the collaborators' nodes reflects the number of connections they have, and the thickness of the lines linking collaborators to files indicates the frequency of commits to those files. The accompanying Table 1 points out a distinct characteristic of two-mode networks: given the disparity between the number of files (178) and developers (7), the connectivity of a file, for instance *urls.py*, is inherently limited to seven, whereas a developer could theoretically modify every file. "Loud Goat," the most active contributor, engaged with 103 files.

Figure 2 - Social network mapping collaborators based on shared files



Centrality measures within the network offer various insights: Degree centrality represents the number of direct links a node has within the network, closeness centrality gauges the average distance of a node to all other nodes, and betweenness centrality reveals a node's frequency as an intermediary in the connections of other nodes, often serving as a conduit or a broker. In this two-mode network, files typically act as such brokers, making the inference of collaboration indirect and left to later, when a collaborator to collaborator social network is built.

The clustering coefficient measures the degree to neighboring nodes are connected to the same nodes. In a two-mode network it will always be zero. For example, in this case collaborators only connect to files. In order for this clustering coefficient be different than zero files would need to connect to files as well.

This social network graph is not yet a vision of collaboration; on the other hand, it is a visual representation that allows for the assessment of top collaborators and those who worked on similar parts of the project, since the graph is laid out according to connections. In this graph, we can see that the student named "Loud Goat" has a degree of 103, which means he has made contributions to that many files. Whereas the one named "Wild Newt" made just one. The eigenvector centrality measures how well a given node is connected to other well-connected nodes. In the case of the two-mode network where collaborators are only connected to files, it can indicate how often one contributes to files that receive many contributions. This measure shows that "Loud Goat" was an important part of the project and worked on parts that were actively being developed by others. This view also highlights the peripheral role of "Wild Newt," who contributed to only one file that wasn't being actively worked on by his colleagues.

Table 1 - SNA parameters for two-mode network connecting collaborators and files

	Degree	Degree Centrality	Betweenness Centrality	Closeness Centrality	Eigenvector Centrality	Clustering Coefficient
Loud Goat	103	0.56	0.58	0.54	0.43	0
Wild Fowl	77	0.42	0.34	0.47	0.37	0

Cute Frog	62	0.34	0.22	0.44	0.31	0
Cute Moth	45	0.24	0.15	0.40	0.23	0
Warm Shrew	24	0.13	0.03	0.37	0.13	0
Dev Wolf	19	0.10	0.05	0.36	0.11	0
urls.py	6	0.03	0.02	0.51	0.12	0
models.py	6	0.03	0.02	0.51	0.12	0
...	...	...	...	...	...	...
013...atualizacao.py	1	0.01	0.00	0.29	0.02	0
Wild Newt	1	0.01	0.00	0.34	0.01	0

The transformation from a two-mode network into a collaborator-centric social network is illustrated in Figure 2. It's important to note that in Figure 2 the "Cute Pig" developer is Github's bot who initialized the project, who was excluded from further analysis . To assess possible collaborations, the two-mode network is further converted into a regular social network connecting the collaborators/students. The strength of the connection between two nodes represents the number of files on which both students worked. This diagram was plotted in a way that developers with strong connections are closer together. The plot is also useful for understanding collaboration from the strength of connections. A cluster of developers seems to have worked intensely on the project, while "Wild Newt" remained distant.

The parameters for this social network of developers are seen in table 2. That table shows that the degree of connection for all collaborators is 5 or 6, meaning that they have shared contributions to project items/files with five or six (all) other members of the project; this measure is reflected in the degree centrality as well. The betweenness centrality is very low because, in general, no collaborator obstructs the connection between two others. The closeness centrality is high because the network, representing the team of seven developers, is well connected. The clustering coefficient is high because all developers are connected to essentially the same set of neighbors.

Social network metrics can help make certain behavioral and organizational aspects more evident. Such as collaborators that are working on several fronts of the project. That does is not necessarily but, but deserves a diagnostic to assess if is a disfunctional group dynamic or some other issue that forced such dynamic.

Table 2 - SNA parameters for collaborator-to-collaborator network

	Degree	Degree Centrality	Betweenness Centrality	Closeness Centrality	Eigenvector Centrality	Clustering Coefficient
Wild Fowl	6	1.0	0.01	1.00	0.39	0.93
Loud Goat	6	1.0	0.01	1.00	0.39	0.93
Cute Moth	6	1.0	0.01	1.00	0.39	0.93
Warm Shrew	5	0.83	0.00	0.86	0.34	1.00

Dev Wolf	6	1.0	0.01	1.00	0.39	0.93
Cute Frog	6	1.0	0.01	1.00	0.39	0.93
Wild Newt	5	0.83	0.00	0.86	0.34	1.00

Similarly, on the other end it can highlight collaborators that shared little project items with colleagues. That may indicate that they worked in isolate because of a disfunctional dynamic in the group. It is worth emphasizing that two collaborators sharing a project item does not necessarily mean they did consciously coordinated their collaboration and actively shared ideas. But intense alternated contribution to some project item can point to a productive dynamic in their partnership. At least items with intense historic of collaboration may help identify parnetships among project members.

CONCLUSION

The SNA metrics used in this work proved useful to characterize a few patterns of behavior. The most evident was that of collaborators that are dettached from the group, such as the "Wild Newt" student, or collaborators that tended to concentrate work, such as "Loud Goat". The metrics applied here could be used while the project is being developed to elucidate the engagement of the collaborators and encourage more productive collaboration. A concern arises if any of these metrics is used during the project or with the objective of grading. There's a risk of individuals prioritizing metrics even to the detriment of the ultimate behavior these metrics indirectly aimed to evaluate (Campbell, 1979) . In an educational context, it's no different, so it's crucial to consider this fact. For instance collaborators might find out how to influence their metrics, for instance commiting to files that their colleagues have just commited to. So any SNA based metric must be complemented with other evidences from the monitoring of the project.

The methodology adopted here can be modified to other types of projects and workers, as long as the interactions with collaborators among themselves and with project deliverables is logged in the form of data. For instance exchanges between people on groupware such as MS Teams and Slack, and participation on documents shared on the cloud or on a corporate network.

This work didn't go into detail into the temporal aspects of the collaboration. The temporal information of the commits is available and could hightlight if collaborators that shared modules of the project alternated their collaboration, adopted a handover aproach with the numerous contributions of a members coming before those of other, or any other kind of dynamics. That must be investigated in a future work.

REFERENCES

ACM. (2013). *Computer Science Curricula 2013:* Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. Association for Computing Machinery.

Andres, H. P. (2012). Technology-Mediated Collaboration, Shared Mental Model and Task Performance. *Journal of Organizational and End User Computing (JOEUC)*, *24*(1), 64–81. https://doi.org/10.4018/joeuc.2012010104

Avila, D. T., Van Petegem, W., Libotton, A. (2021). ASEST framework: A proposal for improving teamwork by making cohesive software engineering student teams. *European Journal of Engineering Education*, *46*(5), 750–764. https://doi.org/10.1080/03043797.2020.1863339

Avila, D. T., Van Petegem, W., Snoeck, M. (2022). Improving Teamwork in Agile Software Engineering Education: The ASEST+ Framework. *IEEE Transactions on Education*, *65*(1), 18–29. https://doi.org/10.1109/TE.2021.3084095

Bacon, D. R., Stewart, K. A., Silver, W. S. (1999). Lessons from the Best and Worst Student Team Experiences: How a Teacher can make the Difference. *Journal of Management Education*, *23*(5), 467–488. https://doi.org/10.1177/105256299902300503

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001). Manifesto for Agile Software Development. *Manifesto for Agile Software Development*. http://www.agilemanifesto.org/

Borgatti, S. P., Everett, M. G. (1997). Network analysis of 2-mode data. *Social Networks*, *19*(3), 243–269. https://doi.org/10.1016/S0378-8733(96)00301-2

Brusoni, S., Prencipe, A., Pavitt, K. (2001). Knowledge Specialization, Organizational Coupling, and the Boundaries of the Firm: Why Do Firms Know More Than They Make? *Administrative Science Quarterly*, *46*(4), Artigo 4. https://doi.org/10.2307/3094825

Campbell, D. T. (1979). Assessing the impact of planned social change. *Evaluation and Program Planning, 2*(1), 67–90. https://doi.org/10.1016/0149-7189(79)90048-X

Chaturvedi, K. K., Sing, V. B., Singh, P. (2013). Tools in Mining Software Repositories. *2013 13th International Conference on Computational Science and Its Applications*, 89–98. https://doi.org/10.1109/ICCSA.2013.22

Costa, G. C. B., Santana, F., Magdaleno, A. M., Werner, C. M. L. (2014). *Monitoring Collaboration in Software Processes Using Social Networks*. Em N. Baloian, F. Burstein, H. Ogata, F. Santoro, G. Zurita (Orgs.), *Collaboration and Technology* (p. 89–96). Springer International Publishing. https://doi.org/10.1007/978-3-319-10166-8_8

Gilbert, E., Karahalios, K. (2009). Using Social Visualization to Motivate Social Production. *IEEE Transactions on Multimedia*, *11*(3), Artigo 3. https://doi.org/10.1109/TMM.2009.2012916

Gousios, G., Kalliamvakou, E., Spinellis, D. (2008). Measuring developer contribution from software repository data. *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 129–132. https://doi.org/10.1145/1370750.1370781

Hira, A., Boehm, B. (2016). Function Point Analysis for Software Maintenance. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–6. https://doi.org/10.1145/2961111.2962613

Hoegl, M., Gemuenden, H. G. (2001). Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence. *Organization Science*, *12*(4), 435–449. https://doi.org/10.1287/orsc.12.4.435.10635

Jorgensen, M., Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering*, *33*(1), Artigo 1. https://doi.org/10.1109/TSE.2007.256943

Katzenbach, J. R., Smith, D. K. (1993, march, 1). The Discipline of Teams. *Harvard Business Review*. https://hbr.org/1993/03/the-discipline-of-teams-2

Kropp, M., Meier, A., Anslow, C., Biddle, R. (2020). Satisfaction and its correlates in agile software development. *Journal of Systems and Software*, *164*, 110544. https://doi.org/10.1016/j.jss.2020.110544

McCabe, T. J. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Moe, N. B., Dingsøyr, T., Dybå, T. (2009). Overcoming Barriers to Self-Management in Software Teams. *IEEE Software*, *26*(6), 20–26. https://doi.org/10.1109/MS.2009.182

Parnas, D. L. (2011). Software Engineering: Multi-person Development of Multi-version Programs. Em C. B. Jones, J. L. Lloyd (Orgs.), *Dependable and Historic Computing:* Essays Dedicated to Brian Randell on the Occasion of His 75th Birthday (p. 413–427). Springer. https://doi.org/10.1007/978-3-642-24541-1_31

Pillai, S. P., Madhukumar, S. D., Radharamanan, T. (2017). Consolidating evidence-based studies in software cost/effort estimation—A tertiary study. *TENCON 2017 - 2017 IEEE Region 10 Conference*, 833–838. https://doi.org/10.1109/TENCON.2017.8227974

Pinheiro, M., Rebelo, T., Lourenço, P. R., Dimas, I. (2023). What drives team learning: Core conditions and paths. *Journal of Workplace Learning*, *35*(2), 146–163. https://doi.org/10.1108/JWL-06-2022-0079

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., Ré, C. (2017). Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, *11*(3), Artigo 3. https://doi.org/10.14778/3157794.3157797

Riquelme, F., Munoz, R., Mac Lean, R., Villarroel, R., Barcelos, T. S., de Albuquerque, V. H. C. (2019). Using multimodal learning analytics to study collaboration on discussion groups. *Universal Access in the Information Society, 18*(3), 633–643. https://doi.org/10.1007/s10209-019-00683-w

Robles, G., Capiluppi, A., Gonzalez-Barahona, J. M., Lundell, B., Gamalielsson, J. (2022). Development effort estimation in free/open source software from activity in version control systems. *Empirical Software Engineering*, *27*(6), Artigo 6. https://doi.org/10.1007/s10664-022-10166-x

Sonnentag, S. (1998). Expertise in professional software design: A process study. *The Journal of Applied Psychology*, *83*(5), 703–715. https://doi.org/10.1037/0021-9010.83.5.703

Sonnentag, S. (2000). Excellent Performance: The Role of Communication and Cooperation Processes. *Applied Psychology*, *49*(3), 483–497. https://doi.org/10.1111/1464-0597.00027

Spadini, D., Aniche, M., Bacchelli, A. (2018). PyDriller: Python framework for mining software repositories. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 908–911. https://doi.org/10.1145/3236024.3264598

Vidoni, M. (2022). A systematic process for Mining Software Repositories: Results from a systematic literature review. *Information and Software Technology*, *144*, 106791. https://doi.org/10.1016/j.infsof.2021.106791

Weimar, E., Nugroho, A., Visser, J., Plaat, A. (2013). Towards high performance software teamwork. *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, 212–215. https://doi.org/10.1145/2460999.2461030

Whitehead, J., Mistrík, I., Grundy, J., van der Hoek, A. (2010). Collaborative Software Engineering: Concepts and Techniques. Em I. Mistrík, J. Grundy, A. Hoek, J. Whitehead (Orgs.), *Collaborative Software Engineering* (p. 1–30). Springer. https://doi.org/10.1007/978-3-642-10294-3_1

World Economic Forum. (2020). *The Future of Jobs Report 2020*. http://www3.weforum.org/docs/WEF_Future_of_Jobs_2020.pdf

Wu, H., Shi, L., Chen, C., Wang, Q., Boehm, B. (2016). Maintenance Effort Estimation for Open Source Software: A Systematic Literature Review. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 32–43. https://doi.org/10.1109/ICSME.2016.87