An overview of software code review tools and the possibility of their application in teaching at the School of Electrical Engineering in Belgrade

Miloš Obradović, Marija Kostić, Balša Knežević, Dražen Drašković, *Member*, IEEE

***Abstract* - The use of version control tools together with the code review techniques is the basis of modern software development. In order to introduce future software engineers to these tools, as well as the process of software development, and to better prepare them for the industry work, the course "Principles of Software Engineering" was formed at the School of Electrical Engineering at the University of Belgrade. Within this course and the team project that students are doing, all the basic stages of the development of a software system are studied. One of the biggest challenges in organizing a practical team project is finding the right tool for code review. This tool should be suitable for educating future engineers, but also enable monitoring of students' progress and evaluation of the work done. This paper presents the basic needs that a software code review tool must meet in order to be suitable for use in education. An analysis of the functionalities of some of the existing code review tools has been given, as well as the possibility of applying these tools in education at the School of Electrical Engineering. The end of the paper presents a proposal for the best way to implement a tool for code review.**

***Index terms* - teaching methodology, program code review, software development.**

I. Introduction

Software has become an indispensable part of our daily lives, and our dependence on software is constantly increasing. An organization's success and reputation depend on its ability to produce and deliver reliable software [1]. Therefore, modern software development requires engineers to not only know how to program properly and effectively but also how to develop good engineering practices to make the codebase healthy and easy to maintain [2]. One of the techniques used in industrial and open-source projects, which aims to control the quality of code added to the codebase, is called code review [3]. The main goal of code review is to improve the readability and maintainability of the codebase. It is a process in which code is reviewed during design and development by someone other than the author. According to [2], a well-designed code review process provides several benefits:

- Allows a reviewer to check the "correctness" of the code change, i.e., is it possible for the change to introduce bugs into the codebase.
- Ensures the code change is comprehensible and understandable to other engineers.
- Enforces consistency across the codebase.
- Psychological and cultural benefits such as promotion of team ownership, validation, and recognition of one's work.
- Enables knowledge sharing.
- Provides a historical record of the code review itself.

Even though it is a widely recommended technique for improving software quality and increasing developers' productivity [4], across the industry, code review is far from the universal practice [2]. Nevertheless, together with the version control systems, the code review process forms the foundation of modern software development.

Future software engineers should be familiar with the tools and processes for version control and code review. Therefore, it is important for engineering students to review each other's source code. However, surprisingly, few engineering courses in universities and colleges include code review activities [5]. The paper [6] provides an overview of the courses that have introduced code review in their practical activities (homework, projects, etc.).

At the School of Electrical Engineering at the University of Belgrade (SEE-UB), the course "Principles of Software Engineering" (PSE) was designed to introduce students to the basic concepts of software engineering. The course covers various aspects of the software life cycle: specification design and user requirements, system design, selection of the most suitable software architecture, implementation, testing, documentation writing, and basic elements of software project management. At the core of this course is a team project in which students go through all phases of the development of a software system. Their activities range from writing basic functional specification and design of the system, to the final, tested and fully functional software product, the so-called release version. The implementation phase is based on creating a web-oriented software system on a monolithic or microservice architecture, using several basic architectural

Miloš Obradović is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: milos.obradovic@etf.bg.ac.rs)

Marija Kostić is with the School of Electrical Engineering and the Innovation Center of the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: marija.kostic@etf.bg.ac.rs), (https://orcid.org/ 0000-0003-4923-3748)

Balša Knežević is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: balsa.knezevic@etf.bg.ac.rs)

Dražen Drašković is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: drazen.draskovic@etf.bg.ac.rs), (https://orcid.org/ 0000-0003-2564-4526)

 ISBN 978-86-7466-930-3

and design patterns. In this school year, students can choose to develop their application using *CodeIgniter* or *Laravel* framework for *PHP*, or *Django* framework for *Python*.

Version control systems and code review process are studied as well. Within the team project, students learn to work in a team and to develop functional software, during the whole semester. Currently, team members are not involved in the code review and code testing process for other team members, so it is the desire of teachers that team members revise each other's program code. Thus, the author of the program code will always receive at least one or two reviews from other members of their team (or optionally members of another team).

In the third phase, students have to formally review the source code that some other team is working on. This phase aims to expand the knowledge and the programming techniques among students, both through what students see in other teams' solutions and through the feedback they receive from colleagues who have reviewed their solution.

For the course activities to be successful, it is necessary to find the best software tools that are available for use at the SEE-UB, which support the version control and code review. Several control version tools that are open-source are quite suitable for use in the course. However, it is much more difficult to find an appropriate system for code review that is publicly available, motivates students to work regularly on their project, and reduces the possibility of some team members avoiding doing their part of the project.

This paper is divided into five sections. The following section describes the process of code review and gives an overview of the functionalities of some popular code review tools. The third section outlines the basic requirements that a code review tool must fulfill to be used for teaching purposes at the SEE-UB. The fourth section presents an analysis of some existing code review tools, and the possibility of their application in teaching. The final section gives a brief conclusion of the results of this work and advice on how to independently implement software code review tools.

II. Analysis of Basic Features of Code Review Tools

Millions of software engineers around the world review source code daily. This process helps in finding and fixing bugs and increasing the quality of the codebase. Code reviews must be done in real-time and in multiple iterations. Thus, the use of the tools in the code review process must be simple and clear enough. Modern code review is characterized by being lightweight. It can be executed at many stages of software development, but it typically takes place before a code change is added to a version control codebase. Fig. 1. represents common steps of a code review process. First, the author creates a code change and submits it for review. Next, developers discuss the change and suggest fixes. This is an iterative process where the author has to deal with the suggested changes. Finally, when one or more reviewers approve the change, it can be added to the codebase. It is also possible to reject a code change [7].

Table I shows the basic features of the code review tools. Only tools that support Git version control system were considered in the analysis. Many of the analyzed tools also support other platforms. Only those functionalities related to the code review process were observed.

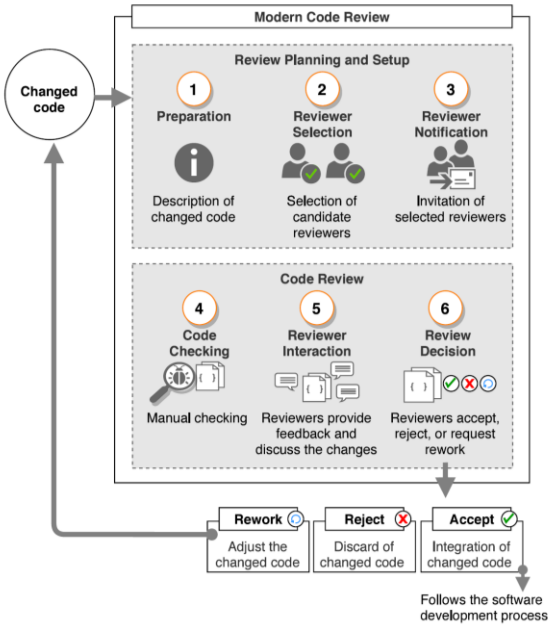

Fig. 1. Common steps of a code review process [8].

III. Necessary Code Review Tool Functionalities

This section presents in detail the basic functionalities of the code review tool that are necessary for the course. Some functionalities are important for the code review and should convey what that process looks like in practice. On the other hand, other functionalities are important for the teaching process and student collaboration on their first team project during their studies. They should motivate students to work in a team and highlight their individual programming and code review abilities. They should also provide better support for documenting a newly developed software system and commenting on different types of artifacts realized in the project phases.

As already noted, this is the first team project in their bachelor's academic studies. It is important that students cooperate well while writing a nice and readable source code, which could be upgraded later with additional modules. Currently, the team project consists of the following phases:

0. Project proposal.
1. Conceptual solution of the project with the basic functional specification.
2. Development of all use cases, one document for each functionality, and realization of the prototype in a tool for prototype development, e.g., Pencil.
3. Formal inspection of the previous phases.
4. Database modeling.
5. UML modeling of the proposed web application.
6. Implementation of a system as a web application.
7. Testing the web application.
8. Final presentation and software documentation.

 ISBN 978-86-7466-930-3

TABLE I OVERVIEW OF CODE REVIEW TOOLS AND THEIR BASIC FEATURES AND FUNCTIONALITIES.

Tool ⇨ Feature ⇩	Bitbucket Server [9]	CodeFlow [10]	Collaborator (previous version: Code Collaborator) [11]	Critique (previous version: Mondrian) [2] [12]	Crucible [13]	Gerrit (fork of Rietveld) [14] [15]	GitHub [16]	GitLab [17]	Space / Upsource [18]	Rhodecode [19]	Review Board [20]
Maintainer	*Attlassian*	*Microsoft*	*SmartBear Software*	*Google*	*Attlassian (former: Cenqua)*	*Google*	*GitHub Inc. (Microsoft)*	*GitLab Inc.*	*Jetbrains*	*RhodeCode*	*reviewboard.org*
Year of origin	2012	2009	2003	2006	2010	2009	2008	2014	2020	2010	2006
Technology stack	*Java*	N/A	N/A	N/A	*Java*	*Java* (1st ver. *Python*)	*Ruby, ECMAScript, Go, C*	*Ruby, Go, Vue.js*	*Java, Kotlin*	*Python (Pylons framework)*	*Python, Django*
License	Proprietary	Proprietary	Proprietary	Proprietary	Proprietary	Apache v2	Proprietary	MIT	Apache v2	AGPL v3	MIT
Open source	no	no	yes	no	no	yes	no	yes	no	yes	yes
Number of users	~10 million	89% of Microsoft employees	~20 000	~50000	N/A	N/A	~73 million	~30 million	N/A	N/A	N/A
Maximal repository memory capacity	4 GB	N/A	N/A	1 TB	N/A	1 TB	100 GB	10 GB	10 GB (free)	N/A	N/A
Maximal file memory capacity	1 GB	N/A	N/A	N/A	N/A	set by admin	2 GB	10 GB	10 GB	10 GB	N/A
Repository privacy	Up to 5 private-free, unlimited public	N/A	N/A	N/A	N/A	Unlimited local repo.	Unlimited public and private	Unlimited	Unlimited	N/A	Private for a fee and public
Solution type	Web-based	Standalone	Web-based	Web-based	Web-based	Web-based	Web-based	Web-based	Standalone, Web-based	Standalone, Web-based	Web-based
Version control systems support	Git, Mercurial	Git	Git, SVN, TFS, Perforce, CVS, ClearCase, RTC	Git, Piper	Git, Mercurial, CVS, Subversion, Perforce	Git	Git	Git	Git	Git, Mercurial, Subversion	Git, Mercurial, CVS, Subversion, Perforce, Bazzar, ClearCase, TFS, IBM Rational ClearCase, HCL
Review document, pictures and diagrams	N/A	N/A	No	Yes	N/A	Yes	No	No	N/A	No	No
Review at character level (prog.code)	No	Yes	No	Yes	No	Yes	No	No	No	No	No
Integration with project management tools	Jira, Trello	N/A	Jira	N/A	Jira	Jira	Jira, Trello	Jira	Trello	Jira, Trello, Redmine, Pivotal	Asana, I Done This, Trello
Integration with other tools	AWS, Crucible, Jenkins, Bamboo, MS Azure, Docker Hub, NPM, Sonar	Visual Studio	Eclipse, Visual Studio, IBM Rational Team Concert, MS Office, Adobe Reader	N/A	Bitbucket	N/A	AWS, Slack, CodeFrash, Semaphore, Asana, Azure, Google Cloud, Heroku, Travis	N/A	IDE (free), Google Calendar (Team), G Suite, Microsoft Office365, Teamcity, Jenkins	Jenkins, Travis CI, TeamCity, Confluence, Slack, HipChat, AppEnlight	Jenkins, Travis CI, Slack, Mattermostm CircleCI, Discord,

In the third phase, the formal inspection process is carried out according to the standard for formal inspection [8] and consists of six activities/steps: planning, product review, inspection meeting, realization of meeting minutes with the defect report and verification of minutes, work on corrections and final follow-up meeting to verify the corrected product. In this phase, one team of students performs a formal inspection to another team by checking all the files and documents made up to that point. The result of this phase are the reports on the formal inspection. These reports are sent to the author team for them to correct all identified shortcomings and defects, and synchronize files from the first two phases of the project. Such formal inspection could be carried out after some lather phases.

 ISBN 978-86-7466-930-3

In the sixth phase, the implementation of the software system, inspection of the source code is necessary, which has been optional so far. This paper aims to decide which software tool would be suitable for students to use in this phase. The functionalities necessary to be part of such a software tool are presented in the following subsections.

A. Availability of tools for use at the SEE-UB

The tool needs to be completely free to use or to have an academic license. Big corporations usually develop their own tools for code review, but do not make them publicly available. Other companies are actively working on the tools for software development, but their solutions are expensive, especially taking into consideration that tool will be used by several hundred students. As the code review techniques are primarily used in the industry, and less as a means of education within academic institutions [5], in the rest of this paper, only open-source tools are considered. Among the most well-known tools that offered packages with academic licenses, the following tools *GitHub*, *Atlassian Bitbucket* and *JetBrains Space* are discussed in Table 1.

B. Roles within the program code review process

The code review largely depends on the participants in the process itself. Currently, there is no universal standard in the software industry that defines exactly which people should check every change in a project [21][22]. Each company defines its own procedures for the reviewer selection process (Fig. 1, step 2). As part of the code review, there are also persons who are only in charge of checking the style of writing the program code, but not for checking the correctness of the functionality of the code [21][22].

For the course PSE there are several required functionalities related to the distribution of different roles:

- Within a single project, all students can participate in the development of the software solution, and all of them are required to review each code change, written by another team member.
- For some changes and monitoring of the program code, it should be possible to add students from other teams as code reviewers. It is necessary to enable manual addition of reviewers or addition based on the programming language/framework. As projects are developed in different programming languages (three different frameworks), it is good that the student, a member of another team, is sufficiently familiar with the syntax of that programming language, in which the authors developed their system. Comments by reviewers more familiar with the code, will be much more useful for the author [8].
- Teaching staff should be automatically added to all the teams. They should be able to just follow students' work without the obligation to review all their code changes, or with possibility to write their own reviews.

C. Possibility of anonymous code review

Research shows that significant discrimination occurs during code review when the authors of the program code or the authors of other documents are known. Discrimination can be based on gender, race, nationality, or age [22]. Additionally, in the school environment, it often happens that students with a lower average grade are afraid to criticize or point out some mistakes of students with higher grades. To motivate students to take the code review process as seriously as possible, as an important part of software development, and to reduce the effects of student discrimination and shyness, the aim is to use a tool that supports anonymous code review. Also, an anonymous review would reduce the possibility of students who know each other making personal arrangements. For example, they could decide to not find many mistakes in each other's code changes to save time they are spending on the project.

D. Ability to check the style of the program code before the code review

Static program code analysis techniques check the structure of the source code without having to execute the program itself. Their objective is to find defects early in the development process. This approach dramatically accelerates the code review process because reviewers can now focus only on the functionality and implementation of the code segment [23]. Some examples of issues that static analyzers can detect are constant expressions that overflow, uninitialized variables, tests that are never run, etc. Next to finding bugs, these tools can help verify that the code is following best practices, style guides, naming conventions, etc. in order to prevent or reduce technical debt [2]. The procedures that would be used within the project may include: writing unit tests and code coverage techniques, by each team member, for their developed parts of the program code, specific styles and precisely defined types of comments, prescribed by course teachers, etc.

As static program code analyzers reduce the cost of software development, many development environments today use them extensively. However, in the case of the course PSE, students have the freedom to choose any available development environment. A static code analyzer that supports *PHP* and *Python* programming languages is required to successfully realize the project within this course.

E. Ability to use tools to review types of files other than the program code

As already mentioned, through the course PSE, students are learning about different phases of software development. Some of the activities students face for the first time are: writing basic functional specifications, developing use case scenarios, testing web applications, and writing appropriate documentation. Next to the source code, students have to work on and produce documents of many different types. Therefore, the tool for code review should support reviewing files that are not source code. The relevant additional types of files are documents, images or specific diagrams, and other multimedia files.

 ISBN 978-86-7466-930-3

F. Ability for the teacher to access basic statistics on tool usage and change the basic code review settings

Teachers should have privileges and advanced functionalities available to them. One of these functionalities is the possibility to set up the code review process. This means that teachers should be able to form teams, control whether anonymous code reviewing is active and determine who are required reviews of some projects. Different statistics about students' work, and contributions should be also available. All this will help in further improving teaching by finding the best setup for the code review process that maximally engages students.

IV. TESTING POPULAR AND AVAILABLE CODE REVIEW TOOLS

This section shows the test results of some popular software code review tools. As explained in subsection A of the third section, for a tool to be used in teaching at the SEE-UB, it must be free for public use. The work of the following tools that meet this condition was tested in detail: *Gerrit*, *GitHub*, and *GitLab*.

Testing of three popular and publicly available tools was performed as follows. The first step is to create a project and make it available on a version control system. After that, it is necessary to do a basic review of the program code within which the tools are tested with basic files for web page structure (*html*), web page layout (*css*), program code files (*js*, *py*, and *php*) and files with commands to work with the database (*sql*). In the next step, the behavior of the code review tool is tested when files that do not contain program code were added to the project. In this step, the behavior of the tools is tested when basic document types (docx and pdf), images (*jpg* and *png*), and diagrams (*uml*) are added to the project. In the last phase of testing the software code review tool, it was checked whether the tool supports other functionalities described in the third chapter.

A. Availability, integration with Git systems, and the possibility to extend functionalities

All three tools have a basic version that is free for public use. *Gerrit* and *Gitlab* have been developed based on open-source code, which opens opportunities to independently upgrade the platform and add new functionalities per personal requirements. *GitHub*, on the other hand, is a closed-source environment, but it has a free version and offers good support for developing open-source projects. The only way to add new features to *GitHub* can be achieved through browser plugins. This is not always a good solution because it adds another item that students must install and use properly. *GitHub* also offers an academic license that brings additional functionality, but at the time of writing, the authors have not been able to go through all the necessary steps to obtain this license, so it was not possible to test its usefulness.

Code review is done through the web interface for all three tools, while support for version control is done through command line. In addition to this, the *GitHub* platform provides the ability to manage versions using the standalone application which can further facilitate the education of students in the use of software development tools. It should also be noted that the *Gerrit* platform has an interface that is not adapted for beginners and has a higher learning curve.

The *Gerrit* platform can be run on a local machine and then all data is stored on that machine. The disadvantage of this approach is that the host machine must always be available and regularly backed up data so that students do not lose their projects due to hardware failure. The advantage of this approach is the ease of adding new functionalities to the tool because the complete code is executed on a local machine.

Gerrit has a slightly different flow control compared to standard *Git* systems. Control has been simplified, which on the one hand may be good for educating new engineers, but on the other hand, it does not follow industry standards, and migrating to another code review git-based system would require adaptation.

B. Working with different file types

As expected, none of the tools had problems working with the program code. All tools provide the ability to comment on each line of program code and set the appropriate status of the comment which indicates that the code is approved or needs to be changed before getting approved.

Problems occur when the tools are used with alternative files such as documents, images, and diagrams. All three tools can display the image, but they are not able to correctly display any type of document. *Gerrit* tool provides a feature to comment on the entire file, while the other two tools do not provide this option. As a result, using *GitHub* and *GitLab*, there is no way to comment on added documents and images.

Diagrams cannot be displayed by any of these tools, but they can display the xml structure that is in the background of this file. The conclusion is that the best way to work with diagrams during software development is to attach an image with each diagram that can be commented.

C. Roles and their permissions within projects and the possibility of anonymous code review

Gerrit allows you to organize users into different groups [14]. It is possible to add users to each group individually, who will have all the authorizations assigned to that group. Using these groups, the teacher can create a project within which they will define groups and their privileges. The teacher also can create subprojects, add students to them and give them predefined group privileges.

GitLab and *GitHub* have predefined roles and access rights within the project. The roles allow the members working on the project to function well when developing software, but do not provide any additional benefits for the teacher role.

No platform has support for anonymous code review. One way to implement an anonymous code review is to make projects publicly visible. Then someone from the other team can look at the program code and submit their remarks externally to the authors of the project. Anonymous review can also be done through web plugins, but one of the problems with using anonymous review through external plugins is that all reviewers must be added to the project,

 ISBN 978-86-7466-930-3

which indicates who the potential reviewers are and reduces the effectiveness of anonymous review. Another problem is relying on the students to use external add-ons correctly, which is not easy to check, so such solutions are not the best.

All three platforms provide basic pre-processor code verification capability before review. As no style guide is currently defined in the course PSE, detailed possibilities of this functionality have not been examined in this paper and are the subject of future research.

Table II maps the functionalities required for teaching the course to the three most popular code review tools, which were publicly available.

TABLE II
ANALYSIS OF FUNCTIONALITIES OF CODE REVIEW TOOLS FOR THE NEEDS OF TEACHING THE COURSE PRINCIPLES OF SOFTWARE ENGINEERING

	Gerrit	*GitLab*	*GitHub*
Public availability	Yes	Yes	Yes
Roles in Code Review	Anonymous Change Owner, Project Owner, Registred User, Custom Gruop	Guest, Reporter, Developer, Maintainer, Owner	Read, Triage, Write, Maintain, Admin
Anonymous code review	No	No	No
Static code analysis	No	Python, PHP	Python
Review files with code	Yes	Yes	Yes
Review document, pictures and diagrams	Can comment on hole file	No	No
Automated user statistics	Available through plugins	Yes	Yes
Project configuration	Yes	Yes	Yes

V. CONCLUSION

This research provides an overview of all the commonly used code review tools in the software industry. Based on this research, the authors recommend the use of *GitLab* software, because it has user-friendly interfaces, is easy to use, has a built-in *Git* version control system, and is based on open-source code. The main disadvantage of using this platform is the limit of up to ten users per project when using the free version.

If you want to develop your code review tool and run it on a local server, then the authors recommend that you start with the *Gerrit* platform, run it on a *Linux* server and add all the features you may need locally. All analyzed tools have some shortcomings and none of them meet all the requirements for application within the course that was analyzed in this paper.

ACKNOWLEDGMENT

This paper is the result of research on the project AVANTES funded by the Science Fund of the Republic of Serbia, within the Program for the development of projects in the field of artificial intelligence. The authors are grateful for the financial support.

REFERENCES

[1] Y.-M. Zhu, “Software Reading Techniques: Twenty Techniques for More Effective Software Review and Inspection,” Solon: Apress, 2016.

[2] T. Winters, T. Manshreck and H. Wright, “Software Engineering at Google Lessons Learned from Programming Over Time,” O'Reilly Media Inc., 2020.

[3] F. A. Ackerman, L. S. buchwald and F. H. Lewski, “Software inspections: An effective verification process.,” *IEEE Software 6,,* vol. 6, no. 3, pp. 31-36, 1989.

[4] Y. K. Wong, “Modern Software Review: Techniques and Technologies,” IRM Press, 2006.

[5] V. Garousi, “Applying Peer Reviews in Software Engineering Education: An Experiment and Lessons Learned,” *IEEE Transactions on Education,* vol. 53, no. 2, pp. 182-193, 2010.

[6] T. D. Indriasari, A. Luxton-Reilly and P. Denny, “A review of peer code review in higher education,” *ACM Transactions on Computing Education (TOCE),* vol. 20, no. 3, pp. 1-25, 2020.

[7] P. C. Rigby and C. Bird, “Convergent Contemporary Software Peer Review Practices,” in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, Saint Petersburg, 2013.

[8] N. Davila and I. Nunes, “A systematic literature review and taxonomy of modern code review,” *Journal of Systems and Software,* vol. 177, 2021.

[8] M. Greiler, “How Code Reviews work at Microsoft,” 22 January 2022. [Online]. Available: https://www.michaelagreiler.com/code-reviews-at-microsoft-how-to-code-review-at-a-large-software-company/.

[9] “BitBucket Code Review,” [Online]. Available: https://bitbucket.org/product/features/code-review. [Accessed: 03 2022].

[10] C. Staff, “CodeFlow: Improving the Code Review Process at Microsoft,” *Communications of the ACM,* vol. 62, no. 2, pp. 36-44, 2019.

[11] “Collaborator - Review Code Together,” [Online]. Available: https://smartbear.com/product/collaborator/overview/. [Accessed: 03 2022].

[12] “Google Mondrian - web-based code review and storage,” [Online]. Available: https://www.niallkennedy.com/blog/2006/11/google-mondrian.html. [Accessed: 11 2021].

[13] “Crucible,” [Online]. Available: https://www.atlassian.com/software/crucible. [Accessed: 3 2022].

[14] “Gerrit - Code Review tool,” [Online]. Available: https://www.gerritcodereview.com/. [Accessed: 12 2021].

[15] L. Milanesio, “Learning Gerrit Code Review,” Birmingham, UK: Packt Publishing Ltd., 2013.

[16] “GitHub - Code Review process,” [Online]. Available: https://github.com/features/code-review. [Accessed: 12 2021].

[17] “GitLab - Code Review Guidelines,” [Online]. Available: https://docs.gitlab.com/ee/development/code_review.html. [Accessed: 12 2021].

[18] “Space - Review Code,” [Online]. Available: https://www.jetbrains.com/help/space/review-code.html. [Accessed: 4 2022].

[19] “RhodeCode - Code Review,” [Online]. Available: https://rhodecode.com/features/productivity. [Accessed: 11 2021].

[20] “Review Board - code and document review tool,” [Online]. Available: https://www.reviewboard.org/. [Accessed: 12 2021].

[21] A. Bosu, M. Greiler and C. Bird, “Characteristics of Useful Code Reviews: An Empirical Study at Microsoft,” in *Proceedings of the International Conference on Mining Software Repositories*, Florence, Italy, 2015.

[22] E. Murphy-Hill, J. Dicker, M. Hodges, C. Egelman, C. Jaspan, L. Cheng, E. Kammer, B. Holtz, M. Jorde, A. Dolan and C. Green, “Engineering Impacts of Anonymous Author Code Review: A Field Experiment,” *IEEE Transactions on Software Engineering,* 2021.

[23] V. Balachandran, “Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation,” in *2013 35th International Conference on Software Engineering (ICSE)*, 2013.