Multidiszciplináris tudományok, 10. kötet. (2020) 3 sz. pp. 277-284 ***https://doi.org/10.35925/j.multi.2020.3.33***

TOOLS, PROCESSES AND FACTORS INFLUENCING OF CODE REVIEW

Nasraldeen Alnor Adam Khleel
PhD student, Institute of Information Science, University of Miskolc
Address: 3515 Miskolc, Miskolc-Egyetemváros, Hungary, e-mail: nasr.alnor@uni-miskolc.hu

Károly Nehéz
associate professor, Institute of Information Science, University of Miskolc
Address: 3515 Miskolc, Miskolc-Egyetemváros, Hungary, e-mail: aitnehez@uni-miskolc.hu

Abstract
Code review is the most effective quality assurance strategy in software development where reviewers aim to identify defects and improve the quality of source code of both commercial and open-source software. Ultimately, the main purpose of code review activities is to produce better software products. Review comments are the building blocks of code review. There are many approaches to conduct reviews and analysis source code such as pair programming, informal inspections, and formal inspections. Reviewers are responsible for providing comments and suggestions to improve the quality of the proposed source code modifications. This work aims to succinctly describe code review process, giving a framework of the tools and factors influencing code review to aid reviewers and authors in the code review stages and choose the suitable code review tool.

Keywords: *code review, the code review process, code review tools, formal code reviews, modern code reviews.*

1. Introduction

Software development projects frequently apply code review phase in their development process [4]. Code Review is an important practice at software companies and open source projects, as it has been proven to enhance software quality, increase awareness, and spread knowledge [15]. The use of analytical methods to examine and revise source codes for error detection was a standard development practice. This process can be accomplished manually and automatically. With automation, where software tools provide help with code review and inspection. By static code analysis or dynamic approach. In the case of static code analysis, source code is analysed without build and execution. [5]. the dynamic method essentially implements the code, executing the program, and dynamically checking for inconsistencies of the presented results [14].

2. Code review

Code review is a well-established software engineering practice [7], where developers submits their code modifications to peers to judge its eligibility to be integrated into the main project codebase [18]. And a software quality assurance activity, that reviews aim to identify defects and improve the quality of the source codes, ultimately, the main purpose of code reviews is to produce a better software product [4]. Using knowledge transfer of design and implementation solutions applied by others [7]. Code

review is the manual or automatic assessment of source code by humans or software, a manual inspection of source code will be done by developers, and this helps in improving the quality of software projects [10]. The person, who does the verification process or reviews the code, called "reviewer" [2]. The primary goal of code reviews is most often to improve the quality of software and reduce defects or enhance the maintainability of source codes [6]. That will be done by carefully inspecting the submitted code for the problem. Code review does not just help build knowledge of application code, but it also enables developers to continuously review changes to the infrastructure [8].

3. Code Review Characteristics and Practice

The purpose of code review characteristics the analysis to understand their relationship with integration decisions. Adequate code reviews can detect bugs, increase productivity, and improve documentation [9]. In practical terms during code review, the developers are performing the role of reviewers and verification process. Reviewers are responsible for providing comments to improve the quality of the proposed corrections. Besides, reviewers also assess whether the corrections are useful to fix problems/defects without breaking the behavior of the system. Then, if the code corrections meet the specified criteria, they can be merged into the main repository [12].

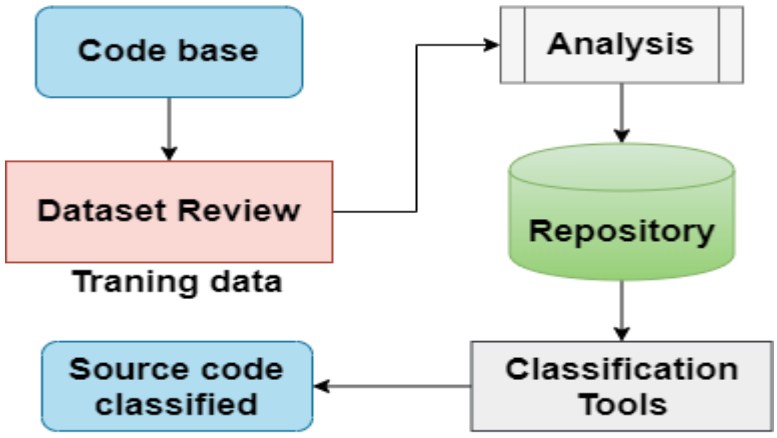

Figure 1. *The architecture of code review and analysis [8]*

4. The code review process

The process of analyzing and verification source code written by another developer on the project to judge whether it is of sufficient quality to be merged into the master project repository [6]. The code review process depending on the components of a review as provided by code review tools [7]. These processes include: code review tools provide an ID and a status for each code review, which are used to track the code change and know whether it has been merged and also allow authors to include a textual description of the code change, to provide reviewers with more information on the rationale and behavior of the change. The second component of a typical code review tool is a view on the technical meta-data on the change under review. This meta-information includes author and committer of the source code modification, commit ID, origin commit ID, and modification ID, that can be used to track the submitted modification over the history of the project, reports about the information on who are the reviewers assigned for the inspection of the submitted code change, lists the source code files

modified in the commit, Finally, the root component for source code review tool and that involves most collaborative parts, reports about the discussion that the author and reviewers are having on the submitted code change. Reviewers can ask clarifications or recommend improvements to the author, who can instead reply to the comments and propose alternative solutions [7]. Figure 2 presents general processes of code review. Code review by the author begins with submitting the code modification (1). The reviewers then verify this modification (2). Based on the project's quality and standards, the reviewers will provide some comments and communicate this to the author (3). If modifying the code meets the project requirements, the reviewers will incorporate the modification directly into the code repository (4). Conversely, if modifying the code does not meets the project requirements, the reviewers reject the modification and the code review is abandoned (5).

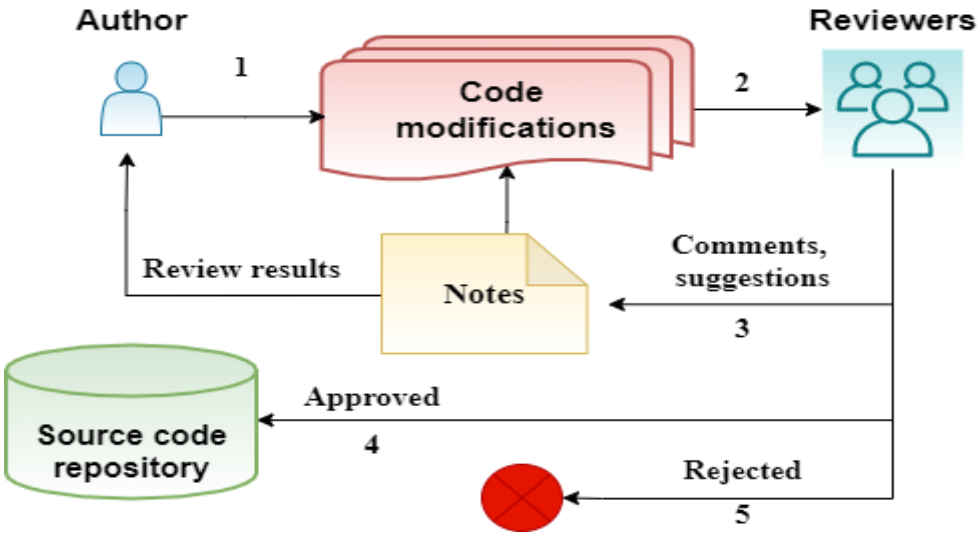

Figure 2. *General code review process [11]*

5. Principal categories of code reviews

Code review is a systematic check of software code. It is purpose is to find errors overlooked in the initial development stage, improving the overall quality of software, and reducing the risk of errors among other benefits [8]. The main building blocks of code reviews and analysis are comments that make the reviewers add and merge their notes and proposal for a modification that the code review author can address. Comments generally can help authors making higher quality modification to the repository, enhance author's development skills and knowledge [6]. There are various approaches to reviews and analysis are such as pair programming, informal inspections, and formal inspections [8]. In general, there are two general approaches to source code reviews: formal inspections (Fagan Inspection) and lightweight source code reviews with an emphasis on efficiency, referred to as modern code reviews (MCR) [2].

5.1. Formal Inspection

Formal code review known as software inspection or Fagan-inspection was first formalized by Fagan in 1976, as manual inspection of source code by developers other than the author for software inspection practice, a structured process for reviewing source code with the single goal of finding defects,

usually conducted by groups of reviewers in extended meetings [11,19]. Formal code review has been an effective quality improvement practice for a long time, and despite their initial success with the many benefits offered by Formal inspections, but relatively high cost and formal requirements have reduced the use it by soft-ware teams adopt them [6]. So, the Fagan inspections method has many limitations that hinder their continuous and popular use across software organizations: they mandate a lot of formal requirements that do not fit well to agile methods, and the waterfall process [2].

5.2. Modern or Contemporary Code Review (MCR)

Modern code review is a collaborative process of code inspection that is often supported by special tools [23]. To ensure that the proposed code modifications are of enough quality and fit the project's progress, the reviewers and authors conduct an asynchronous online discussion [7]. Where the developers read and assess each other's code change before it is integrated into the mainstream line of code towards a release [18]. (MCR) is characterized by little formal requirements, and include tool support, and a strive to make reviews more efficient and less time-consuming. These features allowed many organizations to switch from an occasional to mandatory, continuous employment of reviews [2]. This has shown progress in both industrial and open-source systems [6]. MCR appears a less formal method of conducting source code reviews and analysis, which has been a practice in software engineering for several decades. During source code reviews, developers intend to enhance project quality by fixing errors or making the code easier to be maintained. Developers often use tools that facilitate the code review process [12]. Many factors influence review participation in the MCR process. Where an understanding of these factors helps the team to better manage the code review process [16]. MCR represents a lightweight process, where it is considered (1) informal (in dissimilarity to Fagan-style), (2) tool-based, (3) asynchronous, and (4) focused on inspecting new suggestion source code modification rather than the whole codebase [7, 10, and 19]. Nowadays, many organizations adopt lightweight code review practices to reduce the shortcomings of inspections. There is a clear trend towards using the tools developed to support code review [19]. There are two types of Modern code review tool-base:

5.2.1. Static analysis tools

Static code analysis is an analysis of a computer program that is performed without the actual implementation of the programs built from that software. This means reviewing the source code, and checking compliance with specific rules; basically, static code analysis is performed by two main approaches: self-reviews and third-party reviews, that will be done by the personal software process or team software process [1]. Static analysis tools examine the error code, including those that may lead to software vulnerabilities and issue diagnostic messages ("alerts") indicating the location of the alleged defect in the source code, the nature of the defect, and often include additional contextual information [3]. Many static code analyzers work in different ways. Some static code analyzers work on the source code, while others check the intermediate code and the established libraries. Another difference is the fact that different static analyzers operate on different programming languages [5]. Static code analysis is an activity involving the inspection of source code for quality and security it helps the software developers and testers in detecting and making out several types of flaws. Static code analysis uncovers "hard" bugs before runtime which may be impossible to detect during runtime [14]. There are some advantages and disadvantages of static code analysis, static analysis has many advantages. Where we find that the program to be analyzed does not have to be complete. Also, static analysis can be used early in the software development life cycle. Thus, a report on software quality is received early. This reduces the cost of rework and increases productivity. Test cases do not need to be designed and

"hard" bugs can be detected. The tool has full access to code, that is, it has full access to all possible behaviors of the program. So, it does not need to estimate or understand behavior. Static analysis also has many disadvantages like the production of false positives. Tools like Flaw finder, RATS, ITS4 report many false positives [14].

5.2.2. Dynamic analysis tools

Automated Static Analysis (ASA) can identify common coding problems early in the development process with a tool that automatically checks the source code [5]. (ASA) reports potential source code anomalies, which we call alerts, like a null pointer, dereferences, buffer overflows, and style inconsistencies. Developers inspect each alert to determine if the alert is an indication of an anomaly important enough for the developer to fix [13]. The automated code review software checks the source code to guarantee these source codes confirm with a pre-defined set of principles or best practices. The code review program generally displays a list of warnings and can also provide methods to correct existing problems automatically. Many static code analysis tools can be applied to help automated source code review [17]. Examples to dynamic analysis tools: Data fusion, Graph theory, Machine learning algorisms, Mathematical and statistical models, Dynamic detection tools, Contextual information, and model checking [13].

6. Examples of popular Code Review tools

Bitbucket Server is a Git server and web interface product. It allows users to perform basic Git operations (such as reviewing or merging code) while controlling access to reading and writing the code. The collaborator is a good commercial code and document review. It is used by teams to standardize their review process, reduce defects early, and speed up their development timelines. Crucible is a platform for collaborative code review software. And Web-based code quality tool, it is not open source. It is specially designed for distributed teams and facilitates asynchronous review and comment code. It also integrates with Git and Subversion. Helix TeamHub is a collaboration and hosting tool for code development and tools that support development in Git environments, as well as Apache Subversion and Mercurial. Gerrit is an open-source code, lightweight tool, web-based platform, collaboration tool, and integrates with the Git tool. GitHub is a global platform that provides hosting for a distributed version control system using Git. It provides all the functions of Distributed Release Control and Source Code Management (SCM) for Git as well as adding its features. Where provides many collaboration features like bug tracking, management of task, wiki service and provides access control. GitLab is a web-based platform that provides repositories, and issue-tracking system features. RhodeCode is a platform for open-source code and hosting for firewall source code management. Provides central control over Git, Mercurial, and Subversion repositories within the organization, with shared authorization and authorization management. RhodeCode allows forgery, withdrawal requests, and code reviews via a web interface. Phabricator is an open-source source code, a suite of web-based software development collaboration tools and integrates with version control system tools such as Git, Mercurial, and Subversion. Review Board is a collaborative web and secure code review tool, it is used for code review and document review, Review Board integrates with Bazaar, ClearCase, CVS, Git, Mercurial, Perforce, and Subversion [17].

Table 1. *Comparison of code review tools [17]*

Code Review Software(tools)	Version Control System Tools Integrates	Repository Model	Platform Supported
RhodeCode	Git, Subversion, Mercurial	Distributed and Client-server	Python
Review Board	CVS, Subversion, Git (partial), Mercurial, Bazaar, Perforce, ClearCase, Plastic SCM	Distributed and Client-server	Python
Collaborator	Git, Subversion, Perforce, ClearCase, Mercurial, Rational Team Concert, TFS, Synergy	Distributed and Client-server	Windows, Mac OSX, Linux
GitHub	Git	Distributed	Windows, Mac OSX, Linux
GitLab	Git	Distributed	Ruby on Rails
Phabricator	Git, Subversion, Mercurial	Distributed and Client-server	PHP
Helix TeamHub	Git, Subversion, Mercurial	Distributed and Client-server	Windows, Mac OSX, Linux
Gerrit	Git	Distributed	Java EE
Crucible	CVS, Subversion, Git, Mercurial, Perforce	Distributed and Client-server	Java
Bitbucket Server	Git	Distributed	Java

7. Factors Influencing Code Review

In all software development projects, code review considered an essential part of their development process. Code review aims to enhance the quality of source code modifications made by developers before they are committed to the source code repository. In principle, code review is a transparent process that aims to assess the quality of corrections objectively and promptly; however, in practice, the implementation of this process can be affected by several different factors, technical and non-technical [21]. Most Factors are Influencing Code Review:

Patch Size (LOC): refer to the number of lines of code added or modified in a commit and thus need to be reviewed [22], Patch size under review is the most starting point for any analysis, as it is intuitive that large spots are more difficult to review, and therefore require more time; in fact [21]. Software inspection effectiveness depends on code unit factors such as code size, or functionality [6]. Review Participation (Teams): Teams refer to the many distinct teams associated with the author and invited reviewers [22], Code review is a task that requires the involvement of practitioners to critique new software changes [16]. Locations: includes many distinct geographically distributed development location related to the author and reviewers. Learning of the author: Reviews are expected to trigger the learning of the authors: They get to know their weaknesses, furthermore, they learn new possibilities to solve certain problems. Learning of the reviewer: Reviews are expected to trigger the learning of the reviewers: they gain skills and knowledge about the specific modification and efficiency module [20]. The characteristics of the reviewers and their team, studies suggest that reviewer characteristics can influence review usefulness [6].

8. Conclusion

Code review is a systematic check of the program source code that intended to find errors overlooked in the initial development stage, improving the overall quality of the project and reducing the risk of bugs among other benefits, also can detect bugs, increase productivity, and improve documentation Code review is the manual or automatically assessment of source code by humans or software. This paper helps the researchers in the field of Code review data to know the current tools and process of code review, Factors Influencing Code Review, and how we can selected code review tools depending on version control system tools, Repository model, and Platform supported to identify common coding problems early in the development process.

References

[1] Gomes, I., et al.: An overview on the static code analysis approach in software development, Faculdade de Engenharia da Universidade do Porto, Portugal (2009).

[2] Beller, M., et al.: Modern code reviews in open-source projects: which problems do they fix?, in Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014. 2014. p. 202-211. **https://doi.org/10.1145/2597073.2597082**

[3] Flynn, L., et al.: Prioritizing alerts from multiple static analysis tools, using classification models, in Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies - SQUADE '18. 2018. p. 13-20. **https://doi.org/10.1145/3194095.3194100**

[4] Luxton-Reilly, A., Lewis, A., Plimmer, B.: Comparing sequential and parallel code review techniques for formative feedback, in Proceedings of the 20th Australasian Computing Education Conference on - ACE '18. 2018. p. 45-52. **https://doi.org/10.1145/3160489.3160498**

[5] Jernej, N., Krajnc, A.: Taxonomy of static code analysis tools, The 33rd International Convention MIPRO. IEEE, 2010.

[6] Bosu, A., Greiler, M., Bird, C.: Characteristics of Useful Code Reviews: An Empirical Study at Microsoft, in 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. 2015. p. 146-156. **https://doi.org/10.1109/MSR.2015.21**

[7] Pascarella, L., et al.: Information Needs in Contemporary Code Review. Proceedings of the ACM on Human-Computer Interaction, 2018. 2(CSCW): p. 1-27. **https://doi.org/10.1145/3274404**

[8] Naglot, D., et al.: Code review and analysis using deep learning, International Journal of Rese-

arch and Analytical Reviews (IJRAR). 2019, Volume 6, Issue 2.

[9] Zanaty, F. E., et al.: An empirical study of design discussions in code review, in Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '18. 2018. p. 1-10. **https://doi.org/10.1145/3239235.3239525**

[10] Sadowski, C., et al.: Modern code review, in Proceedings of the 40th International Conference on Software Engineering Software Engineering in Practice - ICSE-SEIP '18. 2018. p. 181-1 **https://doi.org/10.1145/3183519.3183525**

[11] Ebert, F., et al.: Confusion in code reviews: Reasons, impacts, and coping strategies, 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2019. **https://doi.org/10.1109/SANER.2019.8668024**

[12] Panichella, S., et al.: Would static analysis tools help developers with code reviews?, 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015. **https://doi.org/10.1109/SANER.2015.7081826**

[13] Heckman, S., Williams, L.: A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology, 2011. 53(4): p. 363-387. **https://doi.org/10.1016/j.infsof.2010.12.007**

[14] Brar, Hanmeet Kaur, and Puneet Jai Kaur: Comparing detection ratio of three static analysis tools, International Journal of Computer Applications 124.13 (2015). **https://doi.org/10.5120/ijca2015905749**

[15] Bird, C., Carnahan, T., Greiler M.: Lessons Learned from Building and Deploying a Code Review Analytics Platform, in 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. 2015. p. 191-201. **https://doi.org/10.1109/MSR.2015.25**

[16] Thongtanunam, P., et al.: Review participation in modern code review: An empirical study of the Android, Qt, and OpenStack projects (journal-first abstract), in 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2018. p. 475-475. **https://doi.org/10.1109/SANER.2018.8330241**

[17] https://en.wikipedia.org/wiki/Code_review, accessed 2020. January

[18] Asri, I. E., et al.: An empirical study of sentiments in code reviews. Information and Software Technology, 2019. 114: p. 37-54. **https://doi.org/10.1016/j.infsof.2019.06.005**

[19] Bacchelli, A., Bird, C.: Expectations, outcomes, and challenges of modern code review, 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013. **https://doi.org/10.1109/ICSE.2013.6606617**

[20] Baum, T., et al.: Factors influencing code review processes in industry, in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016. 2016. p. 85-96. **https://doi.org/10.1145/2950290.2950323**

[21] Baysal, O., et al.: Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering, 2015. 21(3): p. 932-959. **https://doi.org/10.1007/s10664-015-9366-8**

[22] dos Santos, E. W., Nunes, I.: Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data. Journal of Software Engineering Research and Development, 2018. 6(1). **https://doi.org/10.1186/s40411-018-0058-0**

[23] Ruangwan, S., et al.: The impact of human factors on the participation decision of reviewers in modern code review. Empirical Software Engineering, 2018. 24(2): p. 973-1016. **https://doi.org/10.1007/s10664-018-9646-1**