A Quality Gate Role in a Software Delivery Pipeline

Marko Gluhak and Luka Pavlič

University of Maribor, Faculty of Electrical Engineering and Computer Science, Koroška cesta 46, Maribor, Slovenia

Abstract

This paper is in the area of DevOps, precisely, software delivery pipelines. It outlines the importance of an internal software quality for a long-term success of a software product due to its easier maintenance. By conducting a systematic literature review, we demonstrate, that quality gates in a form of an automatic source code analysis are an underutilized concept. In the paper, we present a case-study, demonstrating an automated quality gate. A prototype pipeline using GitHub Actions and GitLab CI/CD on an existing project, the paper also demonstrates a real-life challenges of implementing a quality gate. They primarily lie in philosophy changes. In addition, software metric thresholds and not straightforward to set.

Keywords

Quality gate, DevOps, CI/CD pipelines, Static code analysis, GitHub Actions, GitLab CI/CD, SonarCloud.

1. Introduction

A current state of the DevOps is constantly evolving [1], adding new testing layer. With each new layer we strive to contribute a more robust product. After several years of evolution in the field, the DevOps pipelines have contributed heavily to the software solutions development cycle enhancements. One of the most important goals of adding robustness to our product, is to facilitate increased confidence to the software release cycle [2]. Companies aspire to build pipelines that allow them to deploy their solutions quickly and with confidence in their stability. With such an aspiration in mind, many fail to pay attention to the product internal quality [3] – namely, the source code quality. Many test cycles merely cover the product external quality or rather, the aspects of the product behavior. The aspect addresses customers perspective, and covers nothing about how our developers perform [4], [5]. An importance of a quality gate on a source code level is understood to be vastly underutilized in the CI/CD pipelines for many software projects [3]. Companies, that can integrate this principle efficiently in their pipeline, might have more competitive advantage as a result [6], [7].

If internal product quality is neglected, we risk accumulation of technical debt [4]. It is a phenomenon which occurs when we sacrifice product quality, usually for the sake of delivering more functionalities. No matter the reason, the short-term outcome is low quality source code. Once we hit a point, where it is under acceptable level, the development can become slower and as a result, cost inefficient. To address this problem, companies adopt approaches such as code reviews, targeted static code analysis, linting etc. While, all of these are valid, they have an issue of relying on human consistency for effectiveness [8]. A key to success lies in establishing widely agreed rules, then are enforced in every development iteration. A quality gate in a CI/CD pipeline can be realized in the form of a static source code analysis. It offers a repeatable and a consistent internal quality analysis after every source code push to a version control system [3].

Our research was motivated by the role of quality gates in practice. This is why we formulated the following research questions:

- **RQ1:** What is the role of the quality gate in the CI/CD pipeline?

SQAMIA 2022: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 11--14, 2022, Novi Sad, Serbia
EMAIL: marko.gluhak@student.um.si (M. Gluhak); luka.pavlic@um.si (L. Pavlic)

- **RQ2:** What are the difficulties, practitioners may face implementing a quality gate?
- **RQ3:** What are the tools, that are most used for implementing a quality gate?

We would like to outline a possible threat and a limitation of our research. In this paper we define the term "quality gate" as an internal software quality check, performed by static source code analysis.

The rest of the paper is structured as follows. In the second chapter we present our research method and details on the systematic literature review. In chapter three, we present the systematic literature review results. The fourth chapter describe a quality gate application into practical case study. In the fifth chapter we focus on answering research questions. And finally, in the sixth chapter, we conclude our work with key insights, ending with acknowledgments and references.

2. Research Method

To research the topic and research questions from the introduction, we've conducted a systematic literature review. Addressing all of the research questions with it, providing a descriptive answer for each. With the review we aim to understand what has currently been done in our research area. To better quantify the third research question, we will also be conducting a counting of tools mentioned in the papers. While not directly correlated to the principle of quality gates, this will give good indication of which tools are more commonly used. It is common that in practice we aim to use already known an established tools in newer concepts. Besides proving reliable in past experiences, they also provide some well needed familiarity when implementing new approaches to software development. The case study is aimed at verifying the findings in practice and enrichen insights of research questions one and two. It will be presented in further detail in the following sections of what we aim to achieve.

2.1. Systematic Literature Review

The systematic literature review was carried out over four of the most popular scientific paper web libraries. They are IEEE, ScienceDirect, SpringerLink and ACM. We've crafted four search queries to address specific research aspects of our proposed principle. The exact search query term and its respective explanation is described in more detail in table 1.

Table 1

Systematic literature review search queries and explanations

ID	Search query	Explanation
1	"quality gate" AND ("devops" OR "software engineering") AND "automation"	Finding connections between quality gates and automation
2	"quality gate" AND ("CI" OR "CD" OR "CI/CD")	Finding papers describing quality gates in CI/CD pipelines
3	"quality gate" AND "automation" AND ("principles" OR "implementation")	Finding studies about implementations and practical examples of quality gate's usage
4	"quality gate" AND "automation" AND ("tool" OR "framework" OR "aid")	Finding the tools that help facilitate the proposed principle.

We've conducted the search for all four queries on the March the 9th in 2022. The libraries yielded 721 papers for our further examination. A tabular representation of papers found by search queries and libraries is presented in table 2.

Table 2
Systematic literature review yielded papers by digital library and search query

ID	Library	Number of papers
1	ACM	10
2	ACM	9
3	ACM	15
4	ACM	19
1	SpringerLink	110
2	SpringerLink	97
3	SpringerLink	117
4	SpringerLink	128
1	IEEE	1
2	IEEE	0
3	IEEE	1
4	IEEE	1
1	ScienceDirect	24
2	ScienceDirect	54
3	ScienceDirect	68
4	ScienceDirect	67

We've then analyzed the papers in a systematic manner, following the inclusion and exclusion criteria. The process of analyzing took 6 phases to complete before the primary research papers were selected. They consisted of the following steps.

- **First phase: digital library search.** With the prepared queries presented in table 1 we queried the digital libraries. We have displayed the results in table 2. We have acquired a total of 721 papers.
- **Second phase: removal of duplicates.** Here, we have conducted a simple check of whether a paper has duplicates in other digital libraries and removed them if so. There are 290 papers left after this step.
- **Third phase: review by title and keywords.** On this step, we have carefully examined the title and keywords of a papers, considering its worthwhileness to our goal of answering the proposed research questions. If the paper seemed useful or we were uncertain of its usefulness it advanced to the next phase. Thirty-six papers remained.
- **Fourth phase: review by abstract.** This phase consisted of us examining and considering the content of the papers abstract. Upon which we concluded whether it was useful or not. If useful or uncertain about its usefulness we added the paper to the next phase. Twenty-three papers remained.
- **Fifth phase: full paper review.** At this stage we fully examined the content of a paper and carefully examined whether it helps us address the research prospects. If so, we added it to the selection of primary paper candidates. If we were uncertain at this stage, we removed the paper from further examination. Nineteen papers remained.
- **Sixth phase: primary research selection.** Here we already had a base of nineteen papers which we added from phase 5. Henceforth we looked at their references, browsed the web for other sources of knowledge. We found six contributions of interest to our research.

After the sixth phase we arrived at the total number of twenty-five primary research sources. We'll analyse them in further detail in the following chapter.

3. Results

In this section we shall present the results of the systematic literature review. Upon its completion we discover that the fields of DevOps, CI/CD pipelines and traditional quality assurance matured in their own regard. The area of static code analysis also gained prevalence over the years. But putting the fields together in the form of an automated quality gate is a more recent phenomenon. Thus, it warrants further study and inspection of how it is and can be further utilized to the benefit of software solution's quality. The papers chosen for a primary base of knowledge is displayed in table 2.

Table 2

Resulting publications of the systematic literature review.

ID	Title	Authors
1	A longitudinal study of static analysis warning evolution and the effects of PMD on software quality in Apache open source projects	A. Trautsch, S. Herbold, and J. Grabowski
2	Agile Development Offers the Chance to Establish Automated Quality Procedures	M. Kösling and A. Poth
3	An empirical characterization of bad practices in continuous integration	F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, and M. di Penta
4	An empirical study on self-admitted technical debt in modern code review	Y. Kashiwa *et al*
5	Challenges and solutions when adopting DevSecOps: A systematic review	R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen
6	Challenges in Adopting Continuous Delivery and DevOps in a Globally Distributed Product Team: A Case Study of a Healthcare Organization	R. K. Gupta, M. Venkatachalapathy, and F. K. Jeberla
7	Comparison of release engineering practices in a large mature company and a startup	E. Laukkanen, M. Paasivaara, J. Itkonen, and C. Lassenius
8	Comparison of Static Analysis Tools for Quality Measurement of RPG Programs	Z. Tòth, L. Vidàcs, and R. Ferenc
9	Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities	T. Amanatidis, N. Mittas, A. Moschou, A. Chatzigeorgiou, A. Ampatzoglou, and L. Angelis
10	Exploration of DevOps testing process capabilities: An ISM and fuzzy TOPSIS analysis	S. Rafi, M. A. Akbar, W. Yu, A. Alsanad, A. Gumaei, and M. U. Sarwar
11	Improving students' programming quality with the continuous inspection process: a social coding perspective	Y. Lu, X. Mao, T. Wang, G. Yin, and Z. Li
12	Introduction of static quality analysis in small- and medium-sized software enterprises: experiences from technology transfer	M. Gleirscher, D. Golubitskiy, M. Irlbeck, and S. Wagner
13	Managing Quality Assurance Challenges of DevOps through Analytics	M. M. Ahmad Ibrahim, S. M. Syed-Mohamad, and M. H. Husin
14	On the Benefit of Automated Static Analysis for Small and Medium-Sized Software Enterprises	M. Gleirscher, D. Golubitskiy, M. Irlbeck, and S. Wagner
15	Strategies to manage quality requirements in agile software development: a multiple case study	P. Karhapää *et al*
16	The 3C Approach for Agile Quality Assurance	A. Janus, R. Dumke, A. Schmietendorf, and J. Jäger
17	The Professional-Grade DevOps Environment	J. Palermo
18	Towards Continuous Software Reliability Testing in DevOps	R. Pietrantuono, A. Bertolino, G. de Angelis, B. Miranda, and S. Russo
19	Towards quality gates in continuous delivery and deployment	G. Schermann, J. Cito, P. Leitner, and H. C. Gall
20	Are Static Analysis Violations Really Fixed? A Closer Look at Realistic Usage of SonarQube	D. Marcilio, R. Bonifacio, E. Monteiro, E. Canedo, W. Luz, and G. Pinto
21	Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges	M. Shahin, M. A. Babar, M. Zahedi, and L. Zhu
22	Carrot and Stick approaches revisited when managing Technical Debt in educational context	Y. Crespo, A. Gonzalez-Escribano, and M. Piattini
23	Code quality & Quality gates Evaluating efficiency and usability of three activities for improving code quality EDAN80-Coaching of programming teams In-depth study	F. Nyberg and J. Skogeby
24	End to End Automation On Cloud with Build Pipeline The case for DevOps in Insurance Industry	M. Soni

To keep in line with our established research questions, we kept closer attention to the tools mentioned in research papers regarding static code analysis. To better justify our main addition to the pipeline in the form of static code analysis we tracked the tools mentioning across the found papers. The results can be observed in table 3.

Table 3
Results of the tools mentioned across papers tracking

Tool	Number of mentions	Paper's IDs
SonarQube	12	1, 2, 3, 8, 9, 11, 14, 17, 20, 22, 23, 24
SonarLint	1	23
PMD	7	1, 9, 12, 14, 16, 20, 23
JArchitect	1	22
Codacy	1	23
FindBugs	7	1, 9, 12, 14, 20, 23, 24
Squore	1	9
CAST	1	9
SonarJ	1	12
Gendarme	1	14
Checkstyle	2	16, 23
NDpenend	1	22

From our tracking activity SonarQube is the most mentioned tool across all our acquired papers in the systematic literature review. It is also worth mentioning that SonarJ and SonarLint found in our papers belong to the same family as SonarQube, but in other forms of integration. Therefore, increasing the number of mentions further than just the twelve. PMD and FindBugs, both share seven respective mentions for a tied second place.

4. Case study: A CI/CD pipeline

To support our theoretical findings from the systematic literature review we conducted a practical example of a CI/CD pipeline and complementing it with a quality gate in the form of automated static code analysis. The goal was to answer the practical aspect of research questions better. To better understand wherein the true difficulty of adopting the principle of automated static code analysis as quality gates in the software solutions delivery pipeline. We set out to prototype a simple yet representative pipeline, which would include the building and testing the pushed code to a repository. Then added a quality gate in the form of static code analysis to it. We present the pipeline design in figure 1. Once we push new code to the GitHub repository building and analysing is done via GitHub Actions. Upon completion of these test our added step of static source code analysis takes place in SonarCloud. If all these steps succeed, we deploy the software solution to production in line with our setup pipeline.

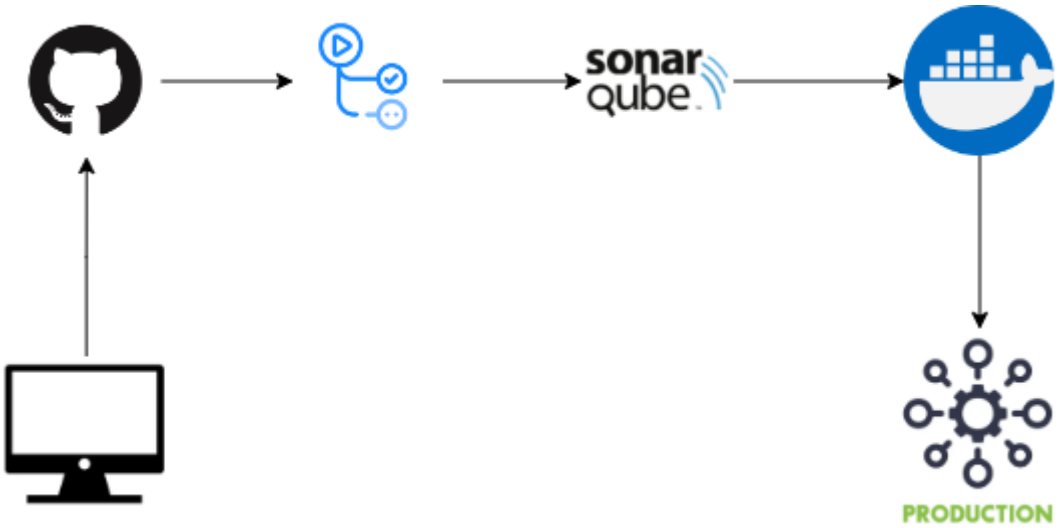

Figure 1: Flow of the software solutions CI/CD pipeline.

We've chosen the following technologies for analyzing our pipeline. All decisions are based on preliminary results of the systematic literature review. We've chosen two providers of CI/CD pipelines, to attempt at better generalizing our findings to the real-world use cases at implementing such ideas to already existing and newly established pipelines.

GitHub is arguably one of the most popular platforms in terms of version control and offers a robust platform for conducting CI/CD tasks in the form of **GitHub Actions**. Although it did not start primarily as a CI/CD tool, it has proved to be effective at this. It requires a YAML specification of a task and then runs it upon the specified ruleset. It should also provide a contrasting candidate for our other choice of CI/CD provider [9]–[11].

GitLab is also a very popular platform for managing version control. The first major difference to GitHub that is owned by Microsoft is that it is open source. Another difference lends itself in terms of its **Gitlab CI/CD** which is purpose built for the task of creating and operating continuous integration and continuous deployment pipelines. Although requiring the same YAML specification to build pipelines it is of a different syntax [12]–[15].

SonarQube is one of the most comprehensive tools for static code analysis. It not only analyzes the quality of said code its technical debt, but also scans for other major security vulnerabilities. Which bridges the gap between DevOps and DevSecOps. Making the results of our research a bit richer and comparative to the real world. It also provides a simple solution for utilization in CI/CD pipelines to allow for easier analysis of procured results [16]. Due to pricing and features offered, we use a cloud variation of SonarQube – **SonarCloud**.

4.1. Project under test

The project used for conducting our case study is a student project, developed by a group of four students over the course of three years, beginning in June of 2019. The project is of a smaller size, counting about six thousand lines of code using the following technologies:

- ReactJS,
- ExpressJS,
- MongoDB,
- ChaiJS,
- GitHub Actions,
- Python.

The project is mostly written in the JavaScript programming language with Python supporting the machine learning aspects of it. The project is deployed on the following link, https://varno-domov.si. Source code can be viewed on GitHub at the following link, https://github.com/MerceneX/BoomMap. The architecture is simple, consisting of three separate parts, communicating with each other

(frontend single page application, backend server and database). All of it is hosted on FERI container services. The domain and DNS are outside providers Domovanje and Free DNS – observable on figure 2.

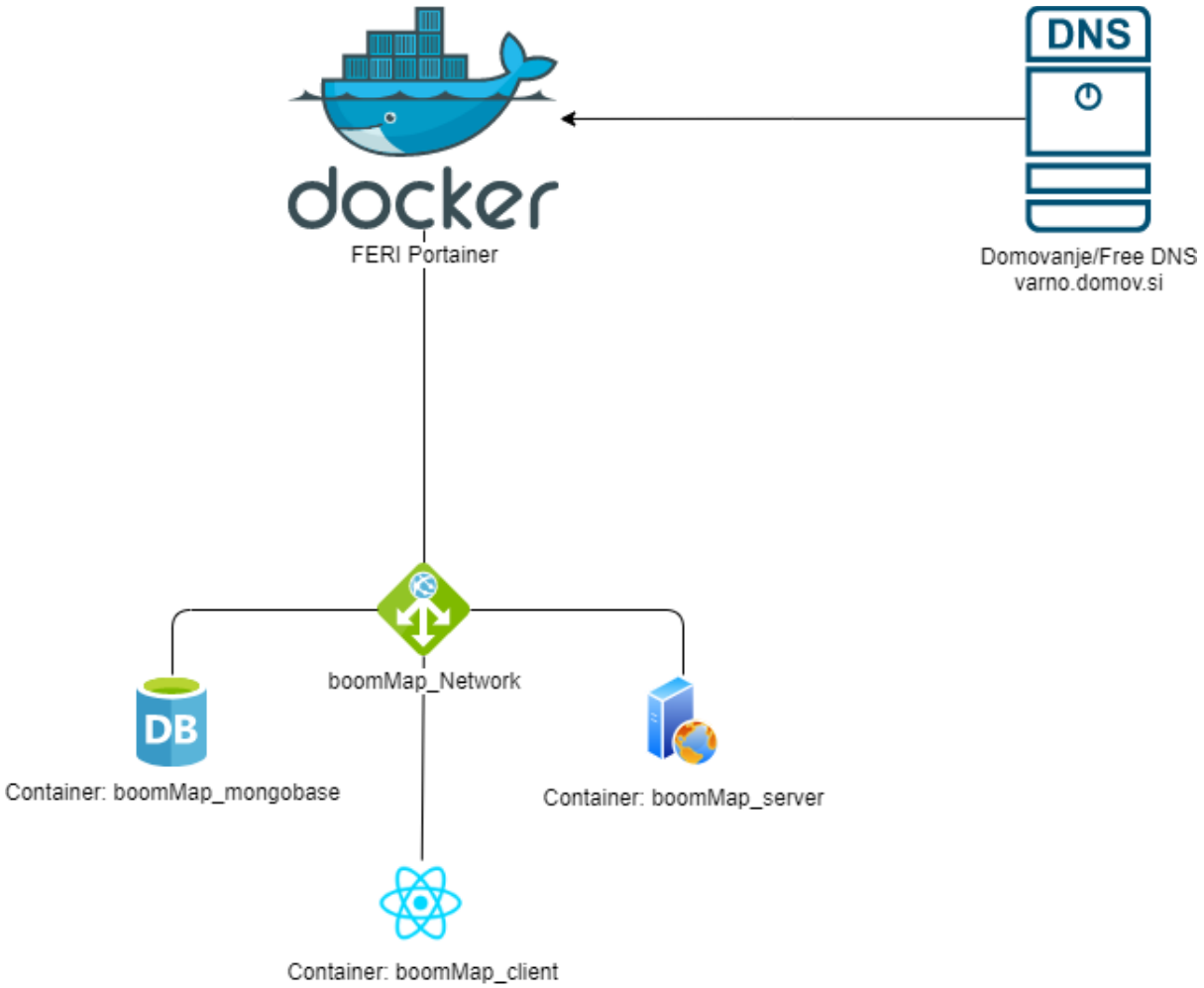

Figure 2: Architecture of the "Varno Domov" project, used for the purposes of the case study.

Current state of the CI/CD pipeline is rather basic in the form of GitHub Actions taking care of three key steps in testing the reliability of the software solution. It is using a matrix strategy to test different Node versions and is being ran on every "push" and "pull request" event. The activities being covered in the steps are as follows:

1. "npm ci" is being ran first, facilitating a clean installation of packets and libraries required for the project to run.
2. "npm run build –if-present" builds our software solution to check if it may be run with the installed libraries and packets
3. "npm run test" takes the code into action and runs the written tests for the code to take place, analyzing if the behavior is as designed.

With these steps we established a simple pipeline with quality checks dependent on each other to succeed. If one is to fail, the pipeline fails, and code is not integrated to the larger codebase.

We are confident that the project with its variable code quality, used platforms and planned additions will provide a good base for our case study and make insights gathered generalizable for further study.

4.2. The Quality gate effects

Upon the addition of the quality gate in the form of SonarCloud's static source code analysis we decided to set a quality gate on new code and not the entire codebase. This is one of the good practices found in the systematic literature review, for it ensures a smoother transition in a new principle's implementation. We can observe the results of the first analysis in figure 3. The added new code was sufficiently written to adhere to the new quality standards.

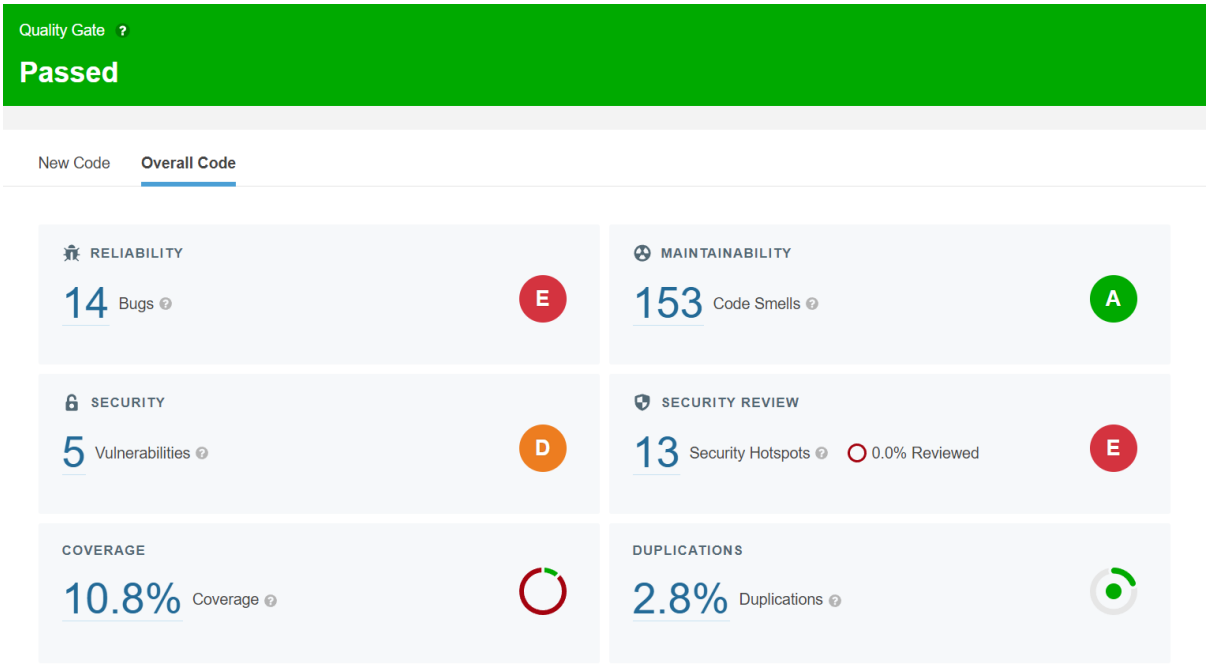

Figure 3: First analysis from SonarCloud and quality gate information

One of the discovered pitfalls can be upon implementation of the quality gate is to set it too strict and on the entire codebase. This can lead to loss of morale among developers when writing code and attempts in cheating the quality gate. The standards should incrementally increase the quality of new code. Thus, increasing the quality of overall code via inevitable alterations in older pieces of code. If quality is set too strict it is almost the same as enforcing an entire rewrite of the codebase. This act, although necessary in some cases, also led many projects to their ruin and the decision of a rewrite should not fall to a single quality gate result.

During our case study, we discovered that if the quality gate is set so that it is feasible to be obeyed, the resulting quality does indeed increase. Although after initial implementation of the quality gate it took, some getting used to, it was manageable and useful. To avoid failings of quality gate an aid like SonarLint might be used to contribute to developer satisfaction.

5. Discussion

To draw conclusions from our research, we begin by answering research questions set in the introduction, to guide our discussion further.

5.1. RQ1: What is the role of the quality gate in the CI/CD pipeline?

Current research on the topic is still fresh. It suggests that the role of the quality gate specified in more detail on a case-by-case basis. That is, when even used. Although companies already have a robust CI/CD pipeline, they do not conduct static code analysis in it, leaving internal quality checks

up to developer or have manual quality gates in place e.g., code reviews. Contrary to intuition, the most common reason for companies not adopting automated quality checks is the worry of quality. Distrust towards tools such as SonarQube, false positives and other wariness towards attitude shifts in assuring quality are all valid concerns.

Although not as commonly practiced of a principle as we had hoped for, there are still two distinguishable types of practitioners that present themselves upon inspection of the papers:

a. Larger companies, which have robust and comprehensive pipelines already in place have a fondness of adding quality checks as possible at every step of the development process. Although assuring quality and having added value it is worth noting that if these checks become too demanding and constraining, they may hinder their own purpose or even the entire development process. Developers may also attempt to look for shortcuts on how to overcome these quality gates.
b. Smaller companies prove to be more successful in practicing the static code analysis quality gate. The concept lends itself better to cloud oriented pipeline services. Worth mentioning is also the fact that it is easier to overcome the attitude barrier towards a new paradigm in projects inception phases rather than down the line when code base becomes less known.

5.2. RQ2: What are the difficulties practitioners may face implementing this principle?

The main difficulty is a psychological one as we have uncovered by our systematic literature review. This notion confirmed by the fact that in our prototypical implementation of a quality gate in a CI/CD pipeline is simple. Regardless of the fact if we are adding it to an existing pipeline or implementing it in a new one. The difference between platforms GitLab CI/CD and GitHub Actions is miniscule enough, to not even warrant a proper comparison. We can draw the decision on which to use up to personal preference of GUI and YAML syntax rules. At least for the purpose of implementing quality gates.

The true difficulties lie in the details and general acceptance of the new paradigm in quality assurance. Most companies list quality as the top concern for not implementing automated quality checks in the form of quality gates. They use code reviews and targeted manual static code analysis. Although a valid concern, because of shortcomings in tools like SonarQube in the form of too many false positives, unrealistic technical debt estimation… It is our finding, that even if these tools do not accurately assess the state of quality, they do serve to increase it.

Setting quality gates on a project-by-project basis appears as the correct way to be. Which also makes sense from a business perspective since a project worth more should be of higher quality. Thus, the challenge is figuring out the correct settings in said quality gates, to reflect the quality expected for that project. And code metrics thresholds are a whole other topic, out of the scope of this paper, but will play a key role in the effectiveness of set quality gates.

5.3. RQ3: Which tools are most used for implementing this principle?

As gathered from our systematic literature review, the most mentioned tool is SonarQube. It supports all the functionalities for a successful static code analysis, it allows project by project configuration of quality gates. That is one of the better practices identified as well, to tailor every project as a case-by-case. Another reason for SonarQube's popularity is without a doubt the ability to integrate the tool at every step of the development process. The second most mentions belong to PMD which is also a generic tool for static code analysis and FindBugs, which is more language focused.

On the topic of platforms that offer CI/CD services there is little to no discussion or mention. Names like Azure and Jenkins come up, with no further elaboration. We assume that larger companies prefer tools that offer more adjustability to their use case an opt for on premise teams for support. Smaller ones prefer to keep it simple, and cloud based.

6. Conclusion

In the end we find ourselves concluding our research with a more thorough understanding of what a quality gate in software delivery pipeline is meant to be – a developer's companion in the search for higher quality. It is there to support decision making and encourage questioning decisions that seem poor in terms of code quality. Although false positives and disparaging estimations of technical debt may undermine this goal, it is however still a boon to use these features. In our systematic literature review we found that the most efficient way to use this principle is with general guidelines and not robust and overly strict rule that fail builds constantly. The principle also aims to improve the mentality shift towards quality assurance and developing cleaner code. Which is still the biggest concern with implementing this principle, that it does not become a hinderance and falls to the side.

7. Acknowledgements

The authors acknowledge financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057)

8. References

[1] A. K K, "DevOps Basics and Variations," *Azure DevOps for Web Developers*, pp. 1–11, 2020, doi: 10.1007/978-1-4842-6412-6_1.

[2] R. Pietrantuono, A. Bertolino, G. de Angelis, B. Miranda, and S. Russo, "Towards continuous software reliability testing in DevOPs," *Proceedings - 2019 IEEE/ACM 14th International Workshop on Automation of Software Test, AST 2019*, pp. 21–27, May 2019, doi: 10.1109/AST.2019.00009.

[3] S. Rafi, M. A. Akbar, W. Yu, A. Alsanad, A. Gumaei, and M. U. Sarwar, "Exploration of DevOps testing process capabilities: An ISM and fuzzy TOPSIS analysis," *Applied Soft Computing*, vol. 116, p. 108377, Feb. 2022, doi: 10.1016/J.ASOC.2021.108377.

[4] Y. Kashiwa *et al.*, "An empirical study on self-admitted technical debt in modern code review," *Information and Software Technology*, vol. 146, p. 106855, Jun. 2022, doi: 10.1016/J.INFSOF.2022.106855.

[5] Y. Crespo, A. Gonzalez-Escribano, and M. Piattini, "Carrot and Stick approaches revisited when managing Technical Debt in an educational context," *Proceedings - 2021 IEEE/ACM International Conference on Technical Debt, TechDebt 2021*, pp. 99–108, May 2021, doi: 10.1109/TECHDEBT52882.2021.00020.

[6] M. Shahin, M. A. Babar, M. Zahedi, and L. Zhu, "Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges," *International Symposium on Empirical Software Engineering and Measurement*, vol. 2017-November, pp. 111–120, Dec. 2017, doi: 10.1109/ESEM.2017.18.

[7] E. Laukkanen, M. Paasivaara, J. Itkonen, and C. Lassenius, "Comparison of release engineering practices in a large mature company and a startup," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3535–3577, Dec. 2018, doi: 10.1007/S10664-018-9616-7/TABLES/14.

[8] A. Janus, R. Dumke, A. Schmietendorf, and J. Jäger, "The 3C approach for Agile Quality Assurance," *2012 3rd International Workshop on Emerging Trends in Software Metrics, WETSoM 2012 - Proceedings*, pp. 9–13, 2012, doi: 10.1109/WETSOM.2012.6226998.

[9] "Features • GitHub Actions · GitHub." https://github.com/features/actions (accessed Feb. 05, 2022).

[10] "GitHub Code & Quality Analysis | GitHub Integration | SonarQube." https://www.sonarqube.org/github-integration/ (accessed Feb. 05, 2022).

[11] "Create a quality gate with GitHub | Clayton Help Center." https://help.clayton.io/en/articles/2152713-create-a-quality-gate-with-github (accessed Feb. 05, 2022).

[12] “GitLab CI/CD | GitLab.” https://docs.gitlab.com/ee/ci/ (accessed May 18, 2022).
[13] “GitLab Code & Quality Analysis | GitLab Integration | SonarQube.” https://www.sonarqube.org/gitlab-integration/ (accessed Feb. 05, 2022).
[14] “CI/CD pipelines | GitLab.” https://docs.gitlab.com/ee/ci/pipelines/ (accessed Feb. 05, 2022).
[15] “Create a quality gate with GitLab | Clayton Help Center.” https://help.clayton.io/en/articles/2871439-create-a-quality-gate-with-gitlab (accessed Feb. 05, 2022).
[16] “Code Quality and Code Security | SonarQube.” https://www.sonarqube.org/ (accessed Feb. 05, 2022).
[17] A. Trautsch, S. Herbold, and J. Grabowski, “A longitudinal study of static analysis warning evolution and the effects of PMD on software quality in Apache open source projects,” *Empirical Software Engineering*, vol. 25, no. 6, pp. 5137–5192, Nov. 2020, doi: 10.1007/S10664-020-09880-1/TABLES/21.
[18] M. Kösling and A. Poth, “Agile Development Offers the Chance to Establish Automated Quality Procedures,” *Communications in Computer and Information Science*, vol. 748, pp. 495–503, 2017, doi: 10.1007/978-3-319-64218-5_40.
[19] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, and M. di Penta, “An empirical characterization of bad practices in continuous integration,” *Empirical Software Engineering*, vol. 25, no. 2, pp. 1095–1135, Mar. 2020, doi: 10.1007/S10664-019-09785-8/TABLES/13.
[20] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, “Challenges and solutions when adopting DevSecOps: A systematic review,” *Information and Software Technology*, vol. 141, p. 106700, Jan. 2022, doi: 10.1016/J.INFSOF.2021.106700.
[21] R. K. Gupta, M. Venkatachalapathy, and F. K. Jeberla, “Challenges in Adopting Continuous Delivery and DevOps in a Globally Distributed Product Team: A Case Study of a Healthcare Organization,” *Proceedings - 2019 ACM/IEEE 14th International Conference on Global Software Engineering, ICGSE 2019*, pp. 30–34, May 2019, doi: 10.1109/ICGSE.2019.00020.
[22] Z. Tòth, L. Vidàcs, and R. Ferenc, “Comparison of Static Analysis Tools for Quality Measurement of RPG Programs,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9159, pp. 177–192, 2015, doi: 10.1007/978-3-319-21413-9_13.
[23] T. Amanatidis, N. Mittas, A. Moschou, A. Chatzigeorgiou, A. Ampatzoglou, and L. Angelis, “Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities,” *Empirical Software Engineering*, vol. 25, no. 5, pp. 4161–4204, Sep. 2020, doi: 10.1007/S10664-020-09869-W/TABLES/8.
[24] Y. Lu, X. Mao, T. Wang, G. Yin, and Z. Li, “Improving students’ programming quality with the continuous inspection process: a social coding perspective,” *Frontiers of Computer Science 2019 14:5*, vol. 14, no. 5, pp. 1–18, Jan. 2020, doi: 10.1007/S11704-019-9023-2.
[25] M. Gleirscher, D. Golubitskiy, M. Irlbeck, and S. Wagner, “Introduction of static quality analysis in small- and medium-sized software enterprises: experiences from technology transfer,” *Software Quality Journal*, vol. 22, no. 3, pp. 499–542, Sep. 2014, doi: 10.1007/S11219-013-9217-Z/TABLES/13.
[26] M. M. Ahmad Ibrahim, S. M. Syed-Mohamad, and M. H. Husin, “Managing quality assurance challenges of Devops through analytics,” *ACM International Conference Proceeding Series*, vol. Part F147956, pp. 194–198, 2019, doi: 10.1145/3316615.3316670.
[27] M. Gleirscher, D. Golubitskiy, M. Irlbeck, and S. Wagner, “On the Benefit of Automated Static Analysis for Small and Medium-Sized Software Enterprises,” *Lecture Notes in Business Information Processing*, vol. 94 LNBIP, pp. 14–38, 2012, doi: 10.1007/978-3-642-27213-4_3.
[28] P. Karhapää *et al.*, “Strategies to manage quality requirements in agile software development: a multiple case study,” *Empirical Software Engineering*, vol. 26, no. 2, pp. 1–59, Mar. 2021, doi: 10.1007/S10664-020-09903-X/FIGURES/6.
[29] J. Palermo, “The Professional-Grade DevOps Environment,” *.NET DevOps for Azure*, pp. 71–94, 2019, doi: 10.1007/978-1-4842-5343-4_3.
[30] G. Schermann, J. Cito, P. Leitner, and H. C. Gall, “Towards quality gates in continuous delivery and deployment,” *IEEE International Conference on Program Comprehension*, vol. 2016-July, Jul. 2016, doi: 10.1109/ICPC.2016.7503737.

[31] D. Marcilio, R. Bonifacio, E. Monteiro, E. Canedo, W. Luz, and G. Pinto, "Are static analysis violations really fixed? a closer look at realistic usage of sonarqube," *IEEE International Conference on Program Comprehension*, vol. 2019-May, pp. 209–219, May 2019, doi: 10.1109/ICPC.2019.00040.

[32] M. Shahin, M. A. Babar, M. Zahedi, and L. Zhu, "Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges," *International Symposium on Empirical Software Engineering and Measurement*, vol. 2017-November, pp. 111–120, Dec. 2017, doi: 10.1109/ESEM.2017.18.

[33] F. Nyberg and J. Skogeby, "Code quality & Quality gates Evaluating efficiency and usability of three activities for improving code quality EDAN80-Coaching of programming teams In-depth study," 2019.

[34] M. Soni, "End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery," *Proceedings - 2015 IEEE International Conference on Cloud Computing in Emerging Markets, CCEM 2015*, pp. 85–89, Mar. 2016, doi: 10.1109/CCEM.2015.29.

[35] L. PAVLIČ, M. OKORN, and M. HERIČKO, "The Use of the Software Metrics in Practice," *SQAMIA*, vol. 2217, 2018.