Software Requirement Specification Document for Automated Code Review and Quality Assurance Tool

Ahmed Elsayed, Mostafa Mohamed, Abdulrahman Osama, Omar Mohamed
Supervised by: Dr Nisreen elsaber , Eng Youssef talaat

January 14, 2024

Table 1: Document version history

Version	Date	Reason for Change
1.0	10-Jan-2024	SRS First version's specifications are defined.
1.1	12-Jan-2024	Functional, non-Functional requirements and Operational Scenarios.
1.2	14-Jan-2024	Edits in Use-Case,Class diagrams.

GitHub: https://github.com/MostafaElSafy/QualiCodeBackEnd
https://github.com/MostafaElSafy/QualicodeFrontend

Contents

Abstract

Our project, QualiCode, is all about making coding a smoother, more enjoyable experience. Imagine a tool that not only helps you write better code but also turns it into a bit of a game and save the time wasted in the manual code reviewing. We want to create a system where developers get instant feedback on their code, tips on how to improve, and be familiar with best practices and even earn points for doing well. It's like a friendly coach and a game rolled into one. With features like leaderboards and achievement badges, we're not just aiming for better code but also a fun and collaborative coding environment. QualiCode is designed for everyone, from professional developers to Students just starting out in software engineering.

1 Introduction

1.1 Purpose of this document

The purpose of this document is to provide a thorough understanding of the requirements for the development of automated code review and quality assurance tools. It acts as a reference for stakeholders such as developers, project managers, team leaders and quality control teams who manage the development process and try to ensure cohesiveness with project goals.

1.2 Scope of this document

This document covers functional, non-functional requirements, Class Diagram and Use-cases diagram for automated code review and quality assurance tools. It describes the features, setbacks and performance expectations that lay the foundation for the development team to build simple and easy-to-use tools that meet the needs of the target users.

1.3 Business Context

In a fast-paced evolving software environment, ensuring code quality is very crucial, Automated code review and quality assurance tools meet this need by automating code base analysis, providing real-time feedback and integrating gamification elements to improve user engagement. This tool is a valuable tool for the development team to improve code quality, reduce errors and overall productivity.

2 Similar Systems

2.1 Academic

2.1.1 Learning based Methods for Code Runtime Complexity Prediction

This paper aims to tackle the challenging task of quantifying the runtime complexity of programming code through machine learning methods. The authors propose a new annotated dataset called CoRCoD that contains program codes with their corresponding runtime complexities. Key contributions of this work include

- Dataset creation: The authors release the CoRCoD dataset, the first public dataset for code runtime complexity. This data set is important for training and testing machine learning models for code complexity prediction.
- Baseline Models: The paper proposes two basic models to predict runtime complexity of code:
  - Manually create objects: This method extracts 14 objects from the Abstract Syntax Tree (AST) of the source code. These features are used to train the classification model using traditional machine learning algorithms.
  - Code Embeddings: The authors use graph2vec to generate code embeddings from ASTs and train a Support Vector Machine (SVM) classifier on these embeddings.
- Analysis and Evaluation: The authors conduct comprehensive analyzes to assess the performance of the original models. The accuracy, precision, and recall of different classification schemes are compared and the results are analyzed to understand the effectiveness of the proposed methods.
- Benefits and Applications: The paper discusses the potential applications of the proposed solution, such as automated coding assignments, IDE-integrated tools for static code analysis, and other areas of real-time data providing code efficiency is valuable. [1]

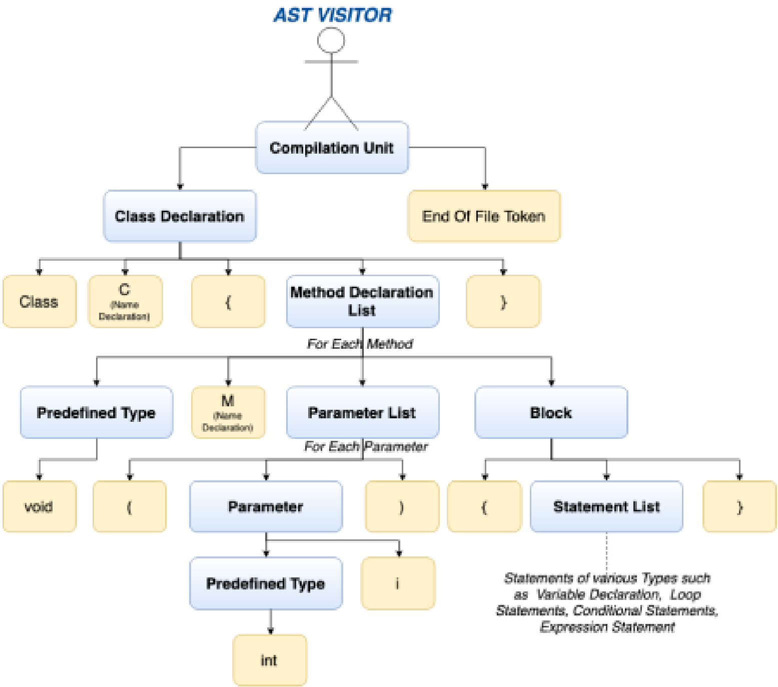


Figure 1: AST Diagram

2.1.2 Deep Learning-based Source Code Complexity Prediction

This paper proposes a deep learning-based method to estimate the worst-case time complicity of an algorithm in source code. The author introduce the CODECOMPLEX dataset, which contains 4,517 real programming codes submitted with corresponding complexity studies written by experts in human programming. The author also present several original programs with their demonstrations for complex code prediction, where classical machine -Learning algorithms as well as state-of-the-art deep learning algorithms such as CodeBERT, GraphCodeBERT, PLBART, and CodeT5. This tool help in estimate the worst-case time complicity of any algorithm and this will apply several benefits :

- Optimization: By analyzing the worst-case time complexity of an algorithm, programmers and algorithm experts can identify areas where the algorithm can be optimized to improve its performance This can result in faster algorithms and effective results, which can be important in applications where speed and efficiency are critical.

- Comparison: Considering the worst-case time complexity of different algorithms can help programmers and system experts to compare the efficiency of different approaches to solving the same problem this can help them to choose an algorithm most effective for a given problem.

- Debugging: Measuring the maximum time complexity of an algorithm can help programmers and algorithms identify performance issues and potential bugs code issues You can perform the analysis

Overall, measuring the most complex time of an algorithm can help programmers and algorithms to optimize their code, choose the most efficient algorithm for a given problem, determine how algorithm will be activated as input data size grows, and performance issues and potential code errors have been identified. [2]

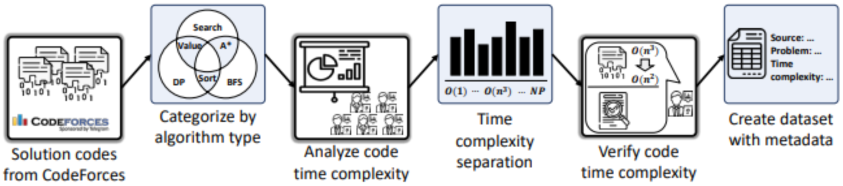

CODEFORCES
Solution codes from CodeForces
Search
Value
A*
DP
Sort
BFS
Categorize by algorithm type
Analyze code time complexity
$O(1)$ ... $O(n^3)$ ... NP
Time complexity separation
$O(n^3)$
$O(n^2)$
Verify code time complexity
Source: ...
Problem: ...
Time complexity: ...
Create dataset with metadata

Figure 2: Overall workflow of CodeComplex dataset creation.

2.1.3 The Benefits of the Coding Standards Enforcement and its Impact on the Developers Coding Behavior-A Case Study on Two Small Projects

This research paper is about the importance of coding standards for software quality and safety, It presents a case study on two Python projects that implemented coding rules and found that producing coding standards can make software quality better and reduce maintenance costs. The paper also discusses the benefits of consistency and reduced workload for testers. in other words , the paper shows that Applying coding standards can help developers write better code , make it easier to maintain and be understandable by the team in software houses We can benefit from this paper that if we make a standard coding style to software houses it will result in a good impact and will make the development teams understand every code easily ,therefore it will save time for the developers and every thing will be clear. [3]

2.2 Business Applications

- EsLint: ESLint is a code analysis tool that identifies potential issues in JavaScript, catches errors early, and is customizable to suit specific styles and needs. It evaluates code using ECMAScript standards and supports configuration of guidelines.

3 System Description

3.1 Problem Statement

In modern software, code quality assurance doesn't only rely on error detection. There are still challenges in maintaining a consistent code style, managing code complexity, and maintaining an environment of continuous evolution among developers. Traditional code review processes often lack efficiency and fail to engage developers. What is needed is a comprehensive automated code review and quality that checks coding styles and measures code complexity according to the rules of the company in coding, Also having a gamified approach to increase developer engagement and motivation. [4]

3.2 System Overview

As shown in Figure 3, that there are two users for the system the first user is the technical lead who will enter the styles and the performance Expectation of the company to the admin view then this data will stored in CRUD page, in the other view for the developer who will submit his code in his interface then the comparison system will compare between the code that the developer entered and the styles and the performance Expectation that the technical lead enter , so it check if the style matches and performance matches and then give it points based on the results for the gamification system and send the feedback back to the interface so the developer could see the review response.

3.2.1 Admin View

In this interface the technical lead choose the styles he want to apply in the check of code style in the check style engine integration and the expectations of the performance of the code so that the machine learning can give the review on the code submitted. Code reviewing and gamification interface In this interface the developer submit his code to review it and receives the feedback for styles and performance expectations from the system and to show his score in the leaderboard with the other developers.[5] [6]

3.2.2 Code Reviewing and Gamified interface

In this interface the developer submit his code to review it and receives the feedback for styles and performance expectations from the system and to show his score in the leaderboard with the other developers.[7] [8]

3.2.3 Check Style Engine Integration

Here we will use api from check style to check styles of the code that will get it from the CRUD page that the technical lead enters and then return a feedback to the developer through the code reviewing interface.[9]

3.2.4 Performance Check

In this part the AST Parsing will used to analysis the code that submitted from the developer and count how many if conditions, for loops, while loops, nested loops and so on, so generate the code features report that will used in the machine learning model.

3.2.5 Machine Learning Model

In this part the machine learning will used to know the complexity of the code by training the model with the dataset we have that will be passed on the AST to analysis the code and calculate the complexity so when the submitted code it knows its complexity and compare it with the expected complexity and give the code reviewing interface the feedback.[10] [11]

3.2.6 CRUD Page

This part will contain the stored styles that the technical lead enter and the performance expectations so that the check style engine and the performance check can compare the submitted code with rules that the technical lead enter.

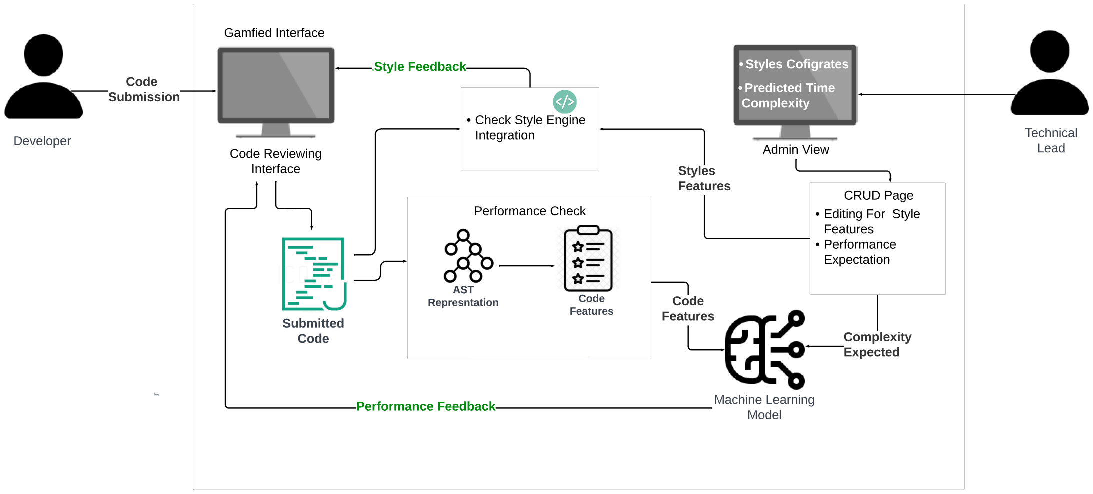


Figure 3: System Overview Diagram.

3.3 System Scope

The Automated Code Review and Quality Assurance Tool aims on the software houses company to help them in saving time, money and effort in the sprint in the reviewing phase. Therefore, in order to achieve that, the system will contain the following :

- Style Feedback: Evaluate and provide feedback on coding style, ensuring compatibility to coding standards and best practices.
- Complexity measurement: Dynamically analyze and measure the runtime complexity of code, which initiates the development of efficient and scalable code.
- Gamification elements: Integrated gamification elements such as scores, leaderboard, badges and challenges to improve user engagement and create a competitive yet collaborative environment.

3.4 System Context

As shown in Figure 4, the developer will submit his code for review and then the system will return for him a feedback about the styles and the complexity of code and increase or decrease his points in leaderboard in the gamification system according to the review, also the technical lead will press on checkboxes of the styles that he want the system to on the submitted code and the performance expectations of the submitted code so the system can review the code of the developer and return feedback.

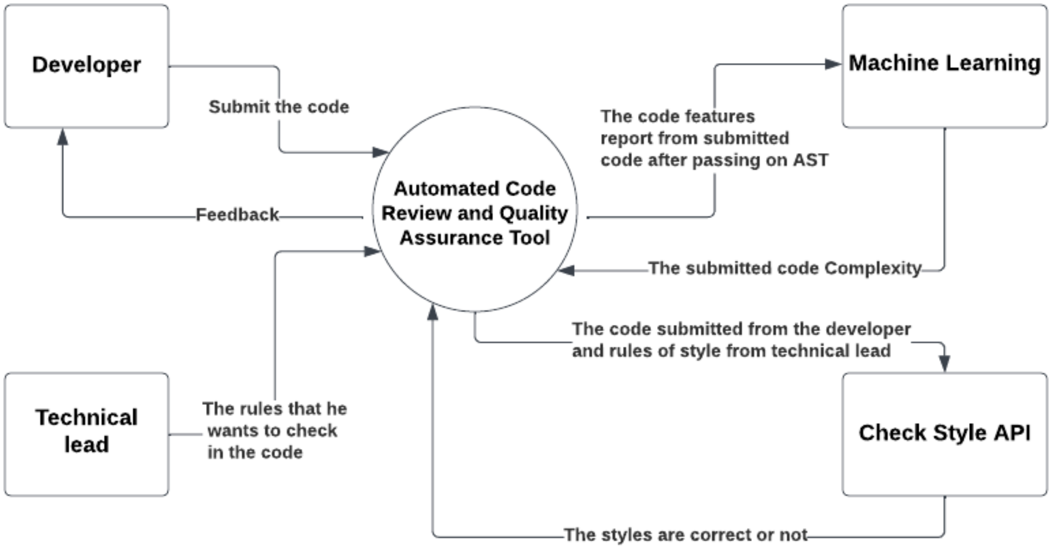

Developer
Submit the code
Feedback
Automated Code Review and Quality Assurance Tool
Machine Learning
The code features report from submitted code after passing on AST
The submitted code Complexity
The code submitted from the developer and rules of style from technical lead
Check Style API
Technical lead
The rules that he wants to check in the code
The styles are correct or not

Figure 4: Context Diagram

3.5 Objectives

The main goals of the automated code review and quality tool are:

- Automated feedback: to provide developers with detailed feedback on code style, and code complexity during the development process.
- Engagement: Integrated elements of gamification approach to motivate developers, encouraging a proactive approach to improving code quality.
- Efficiency and scalability: Identify areas to improve code efficiency and scalability using dynamic complexity analysis.

3.6 User Characteristics

The main users of the tool are programmers, project managers and quality control teams. Developers range from students who are eager to learn to experienced software houses employees, and the tool is suitable for different skill levels, providing guidance and challenges for all user profiles. Project managers and quality assurance teams benefit from the summary reports and insights generated by the tool to make informed decisions about code quality and project progress.

4 Functional Requirements

4.1 System Functions

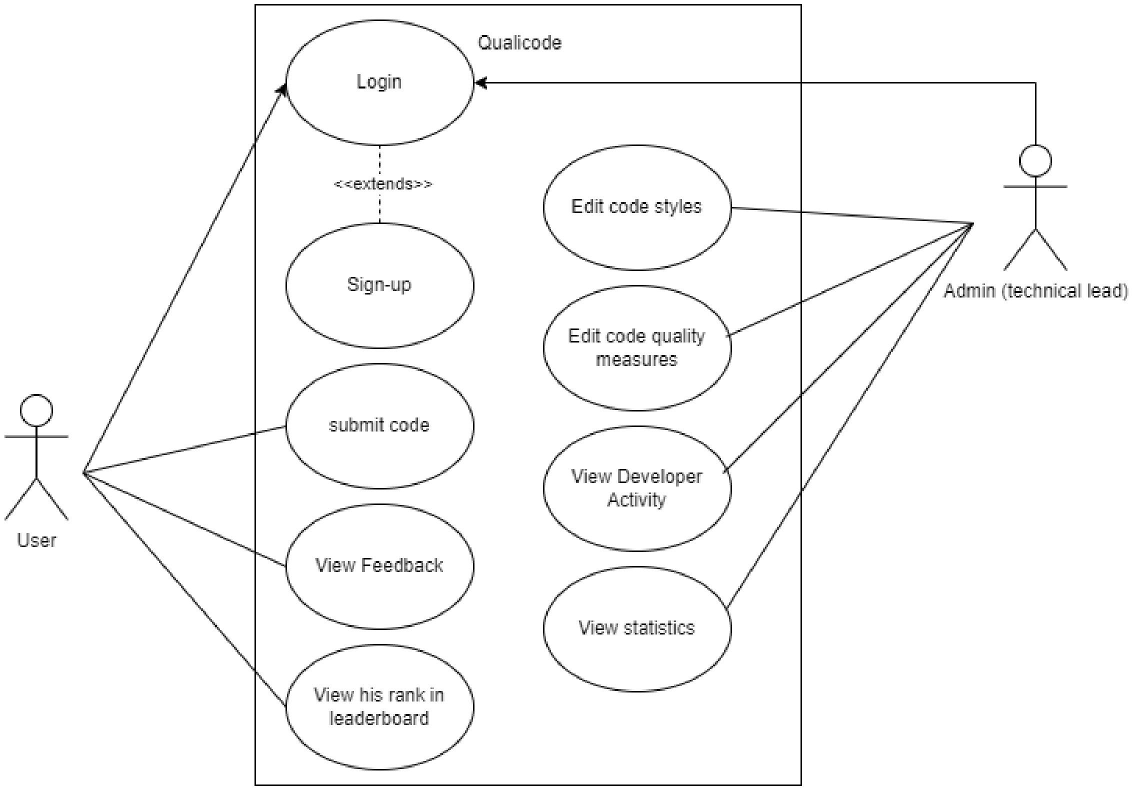


Figure 5: use-case Diagram

4.1.1 User Management

ID:01 Sign up

- Users shall be able to create accounts by providing necessary information.
- The system shall verify the uniqueness of usernames and email addresses during registration.

ID:02 Login

- Users must log in with valid credentials (username/email and password) to access the system.

ID:03 User Roles

- The system shall define different user roles, including developer and admin.
- Admins shall have additional privileges for configuring the system.

ID:04 User Profile Management

- This function enables users to manage their profiles, providing a personalized experience and ensuring accurate representation within the system.

4.1.2 Code Review Functionality

ID:05 Code Submission

- Users shall be able to submit code for review through the user interface.
- The system shall accept code files in common programming languages.

ID:06 Automated Code Reviews

- Implement an automated code review system to analyze submitted code for style and performance issues.
- The system shall integrate with external tools like the ESLint/Checkstyle engine for enhanced code analysis.

ID:07 Feedback and Comments

- Users shall be able to view feedback and comments on their code reviews.

4.1.3 Gamification Elements

ID:08 Leaderboards

- The system shall maintain a dynamic leaderboard ranking developers based on code quality.
- Points shall be awarded for clean code, adherence to best practices, and prompt issue resolution.

ID:09 Achievements and Badges

- Implement an achievement system with badges for milestones and consistent high-quality code.
- Example badges include "Best Dev" for top performers.

4.1.4 Rewards and Recognition

ID:10 Recognition System

- Recognize top-performing developers with tangible rewards such as gift cards, additional vacation days, or other perks.

4.1.5 Admin Functionality

ID:11 Admin Dashboard

- Admins shall have access to a dashboard for system configuration and monitoring.
- The dashboard shall provide insights into user activity, code quality measures, and leaderboard statistics.

ID:12 Configuration Settings

- Admins shall be able to configure system settings, including code analysis rules, reward criteria, and leaderboard calculations.

4.1.6 System Performance

ID:13 Performance Improvement

- The system shall aim to improve the development cycle by a specified percentage (e.g., 30 percentage) in agile teams.
- Regular monitoring and optimization of system performance shall be conducted to ensure efficient code analysis and responsiveness.

4.2 Detailed Functional Specification

Table 2: Automated Code Reviews

Function Name	Automated Code Reviews
Code	ID:06
Priority	High
Critical Level	Critical
Description	This function involves implementing an automated code review system to analyze submitted code for style and performance issues. The system integrates with external tools like the ESLint/Checkstyle engine for enhanced code style and others for performance.
Input	Code files submitted for review. Parameters for code analysis (e.g., style rules, performance thresholds).
Output	Analysis report highlighting style and performance issues. Code review result.
Pre-condition	Valid code files must be submitted. External tools (e.g., ESLint/Checkstyle) are properly configured.
Post-condition	Analysis results and suggestions are available for the user and reviewers.
Dependency	Dependent on the availability and proper configuration of external code analysis tools. Relies on accurate code submissions from users.
Risk	Inaccurate analysis results may lead to incorrect code recommendations. Dependency on external tools may introduce compatibility issues.

Table 3: Configuration Settings

Function Name	Configuration Settings
Code	ID:12
Priority	High
Critical Level	Critical
Description	This function allows administrators to configure system settings, including code analysis rules, reward criteria, and leaderboard calculations.
Input	Admin credentials for authentication. Parameters for configuring code analysis, reward criteria, and leaderboard calculations.
Output	Confirmation of successful configuration. Updated system settings.
Pre-condition	Valid admin credentials are provided. The system is in an operational state.
Post-condition	New configurations are applied to the system. Configurations are stored persistently.
Dependency	Dependent on admin privileges. Influences code analysis, reward system, and leaderboard functionalities.
Risk	Incorrect configurations may impact code analysis accuracy. Misconfigurations could affect user rewards and leaderboard rankings.

Table 4: Leaderboards

Function Name	Leaderboards
Code	ID:08
Priority	Medium
Critical Level	Moderate
Description	This function involves maintaining a dynamic leaderboard ranking developers based on code quality. Points are awarded for clean code, adherence to best practices, and prompt issue resolution.
Input	User performance data (e.g., code quality, issue resolution time). Parameters for leaderboard calculations.
Output	Updated leaderboard rankings. Points awarded to developers.
Pre-condition	User performance data is accurate and up-to-date. The system is in an operational state.
Post-condition	Leaderboard reflects the latest user rankings. Points are attributed to developers based on their performance.
Dependency	Dependent on accurate user performance data. Influenced by configuration settings for points allocation.
Risk	Inaccuracies in user performance data may affect leaderboard accuracy. Incorrect configuration of points allocation may impact the fairness of rankings.

Table 5: User Profile Management

Function Name	User Profile Management
Code	ID:04
Priority	Medium
Critical Level	Moderate
Description	This function enables users to manage their profiles, providing a personalized experience and ensuring accurate representation within the system.
Input	User details (e.g., name, bio, profile picture). Preferences (e.g., notification settings). Password change requests.
Output	Confirmation of successful profile updates. Visual representation of the updated profile.
Pre-condition	Valid user authentication. Access to the user profile management interface.
Post-condition	Updated user profile details are stored in the system. Changes are reflected in the user interface.
Dependency	Dependent on user authentication. Influences the user interface to ensure a personalized experience.
Risk	Incorrect handling of profile data may lead to privacy concerns. Inadequate security measures may result in unauthorized profile access.

5 Design Constraints

5.1 Standards Compliance

The user must have working computer (pc) or (labtop) connected to internet to use our tool (website) and examine the code ,and if the ram from 4 GB to up will be better to have good quality and experience .

5.2 Hardware Limitations

The user only must have pc or laptop.

5.3 Other Constraints as appropriate

It is essential to have stable internet connection to use QUALICODE Tool.

6 Non-functional Requirements

6.1 Security

Access to the tool must be secured through robust user authentication mechanisms, supporting multi-factor authentication for enhanced security.

6.2 Reliability

- The system should maintain at least 99 percentage availability at all times.
- In the event of a server failure, the tool should have mechanisms in place to ensure maintain of user data and the ability to resume normal operation.

6.3 Performance

The system should provide real-time feedback to developers, with a maximum response time for error detection and coding style feedback.

6.4 Maintainability

- The source code of the tool should be well-documented, providing clear and concise documentation for developers contributing to the project or extending its functionality.
- The system should support updates and patches, ensuring minimal downtime and user disruption during maintenance activities.

6.5 Compatibility

The tool should be compatible with major web browsers, including Chrome, Firefox, Safari, and Edge.

6.6 Usability

- The user interface should be engaging yet simple, with clear navigation and visually appealing elements. It should be accessible to users with varying levels of coding experience.
- The tool should have a short learning curve, allowing users to understand and utilize its features efficiently within a maximum of one hour of use.

6.7 Scalability

The tool must scale efficiently to tolerate codebases of different sizes, supporting projects ranging from small to large-scale software systems.

7 Data Design

Our dataset is code python we have it from kaggle and the type of files .txt this codes have number loops,ifs, and nested loops this dataset we will use it to train our model to calculate complexity and this codes make functions like rearrange array, find maximum value in array and so on, it makes some functions with its main that uses loops, ifs, and nested loops.

8 Preliminary Object-Oriented Domain Analysis

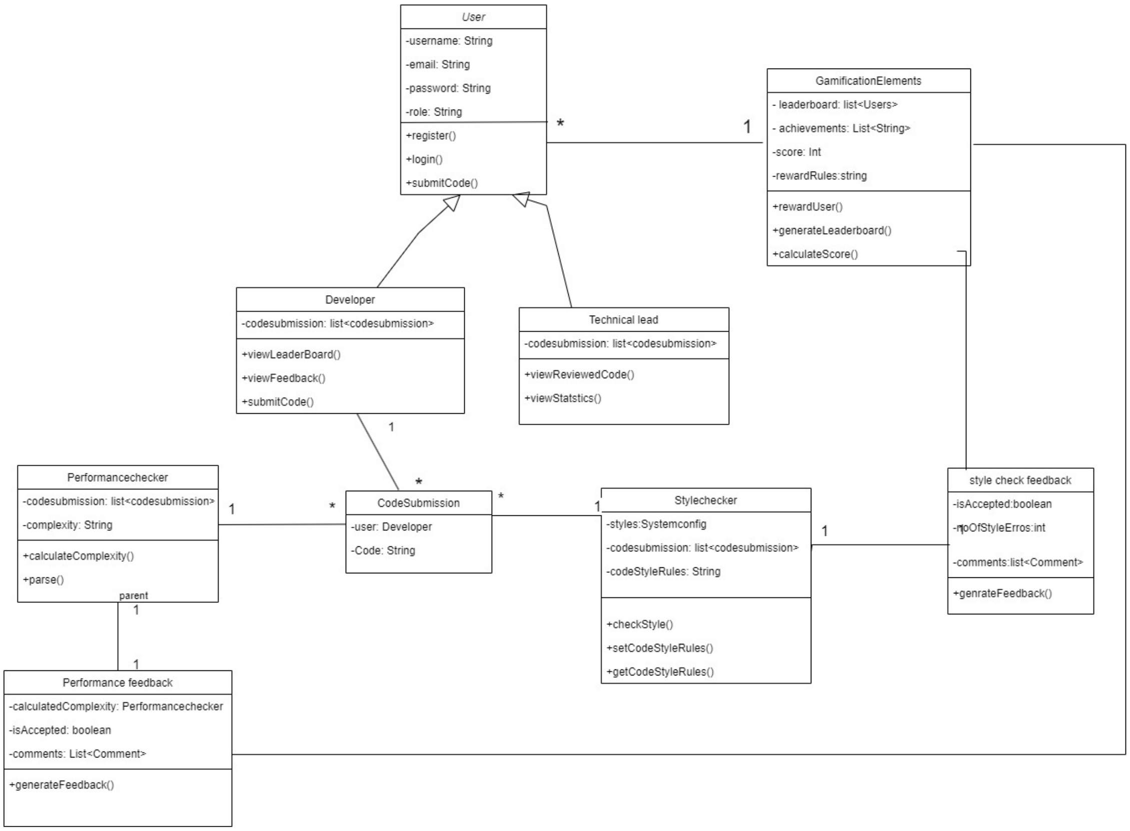


Figure 6: Class Diagram

9 Operational Scenarios

9.1 Scenario (1): User Registration

A new developer joins the team and needs to register on the system. The system should allow simple user registration, guiding the user through an easy process that introduces key features and gamification elements.

9.2 Scenario (2): Code Submission and Analysis

The developer starts by logging into the system and then submits a code for review. The system should efficiently analyze the code, providing feedback on coding style, and runtime complexity. The developer receives a gamified score and badges based on the quality of the submission.

9.3 Scenario (3): Performance Analysis and Optimization

The developer starts by logging into the system and he is working on a program and wants measure runtime complexity of code. The system dynamically analyzes the code. The developer then refines the code based on feedback.

9.4 Scenario (4): User Engagement Challenges

The system initiates a coding challenge or competition among developers to refactor a specific portion of the codebase. Participants receive gamified rewards and recognition based on their performance and the efficiency of their code.

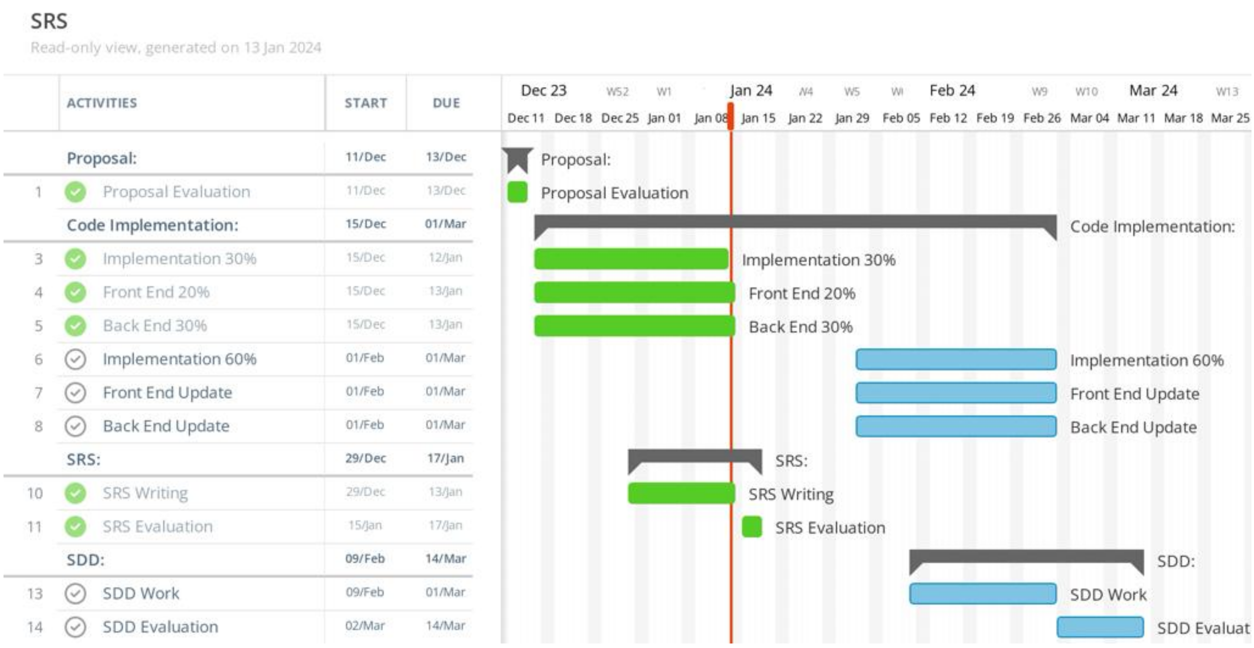
9.5 Scenario (5): Reports and Analytics for Project Managers

The technical lead starts by logging into the system and assess the overall code quality and progress of a development project. The system generates detailed reports and analytics, highlighting areas of improvement, trends, and individual developer contributions. their performance and the efficiency of their code.

9.6 Scenario (6): Putting/update the check styles roles and performance expectation

The technical lead starts by logging into the system and then choose the checkboxes that he want to check on the developers code and the expectation of the performance of the code.

10 Project Plan



SRS

Read-only view, generated on 13 Jan 2024

		ACTIVITIES	START	DUE
		Proposal:	**11/Dec**	**13/Dec**
1	✓	Proposal Evaluation	11/Dec	13/Dec
		Code Implementation:	**15/Dec**	**01/Mar**
3	✓	Implementation 30%	15/Dec	12/Jan
4	✓	Front End 20%	15/Dec	13/Jan
5	✓	Back End 30%	15/Dec	13/Jan
6	○	Implementation 60%	01/Feb	01/Mar
7	○	Front End Update	01/Feb	01/Mar
8	○	Back End Update	01/Feb	01/Mar
		SRS:	**29/Dec**	**17/Jan**
10	✓	SRS Writing	29/Dec	13/Jan
11	✓	SRS Evaluation	15/Jan	17/Jan
		SDD:	**09/Feb**	**14/Mar**
13	○	SDD Work	09/Feb	01/Mar
14	○	SDD Evaluation	02/Mar	14/Mar

Figure 7: Time Plan

11 Appendices

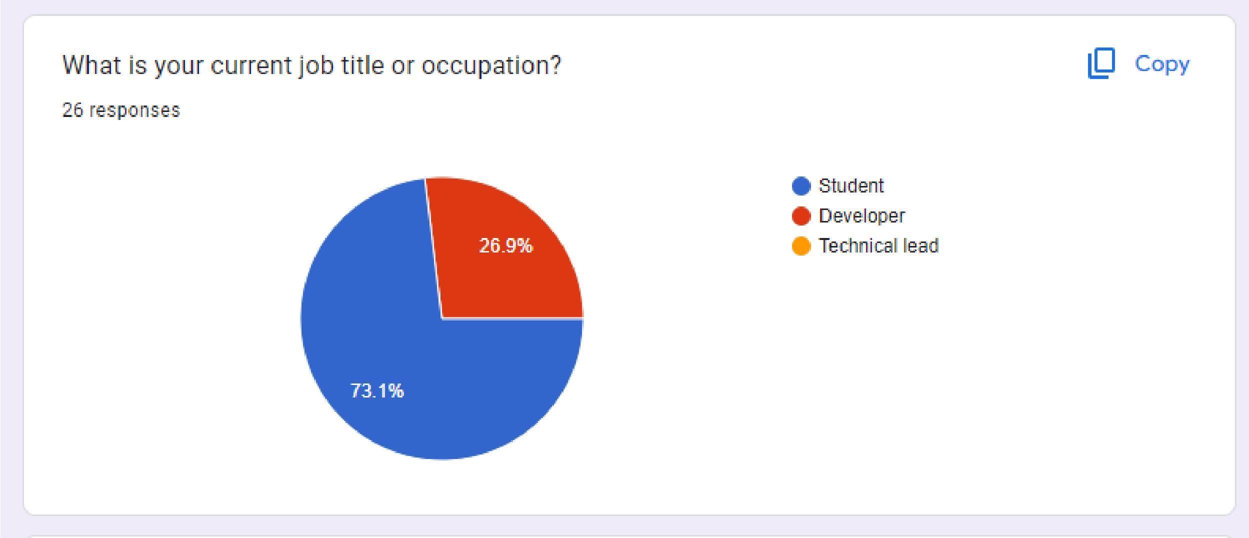
11.1 Definitions, Acronyms, Abbreviations

Table 6: Definitions

Term	Stands for
AST	Abstract syntax tree
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
SVM	Support Vector Machine

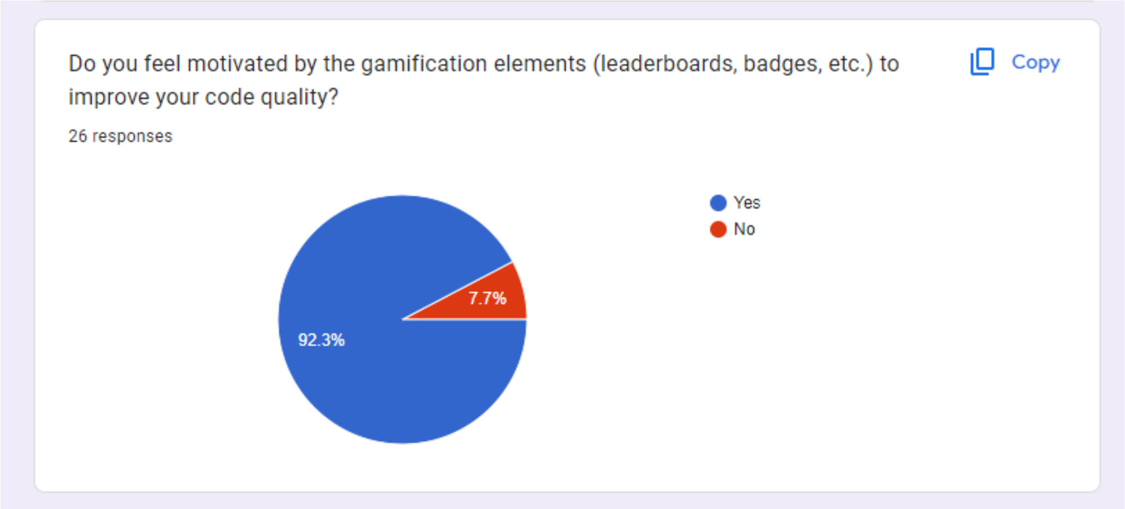
11.2 Supportive Documents

This is surveys that we asked developers, project managers and technical leads who could be interested in our tool:



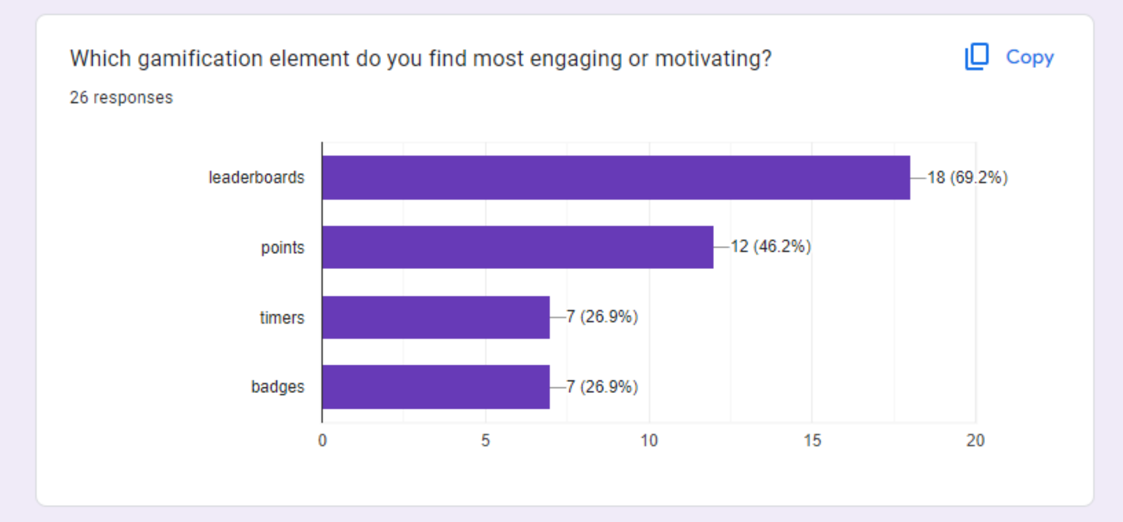
What is your current job title or occupation?

26 responses



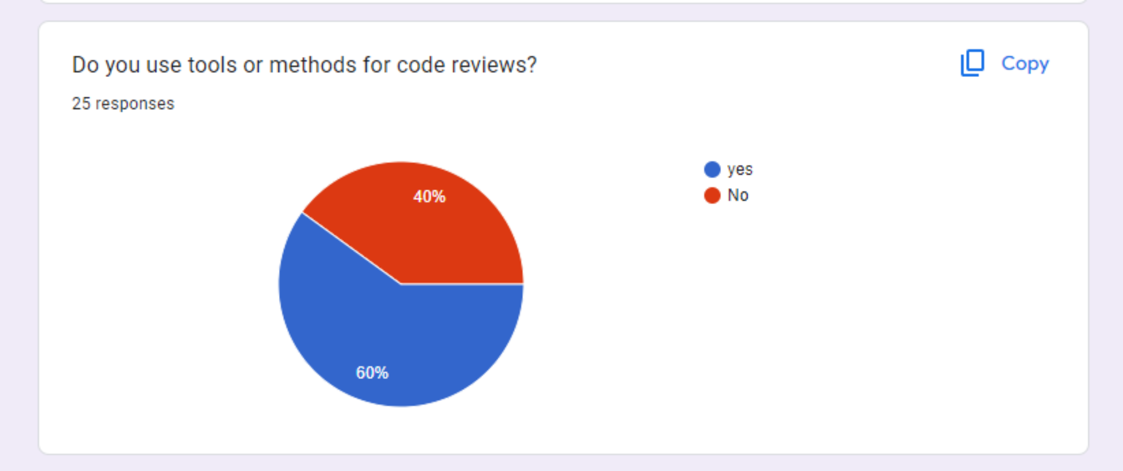
Do you feel motivated by the gamification elements (leaderboards, badges, etc.) to improve your code quality?

26 responses



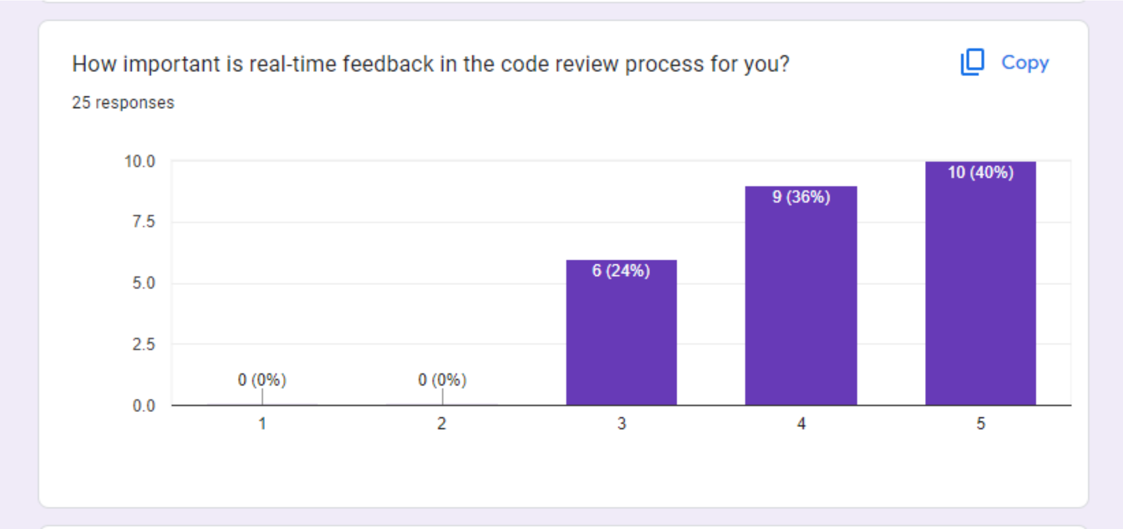
Which gamification element do you find most engaging or motivating?

26 responses



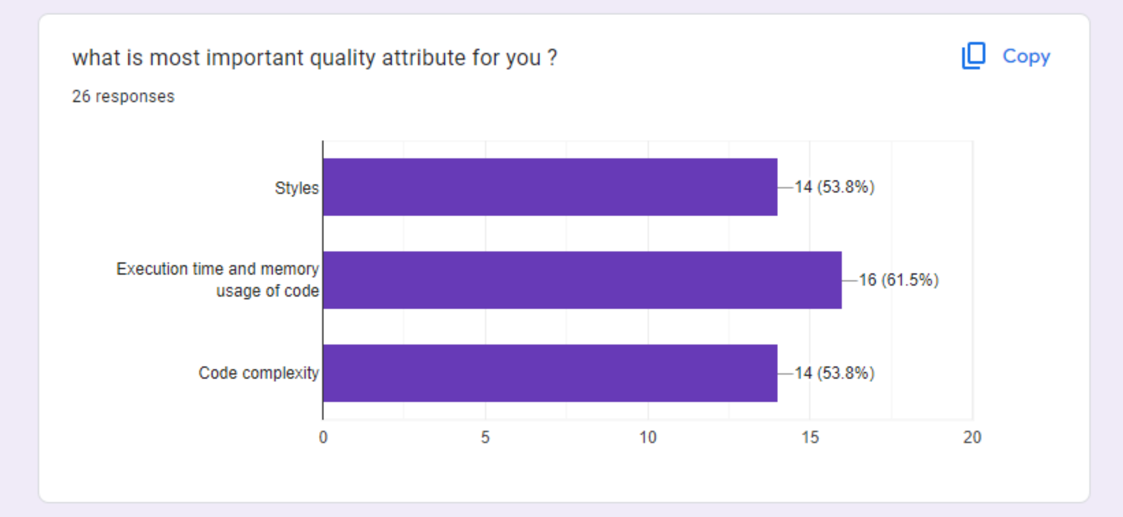
Do you use tools or methods for code reviews?

25 responses



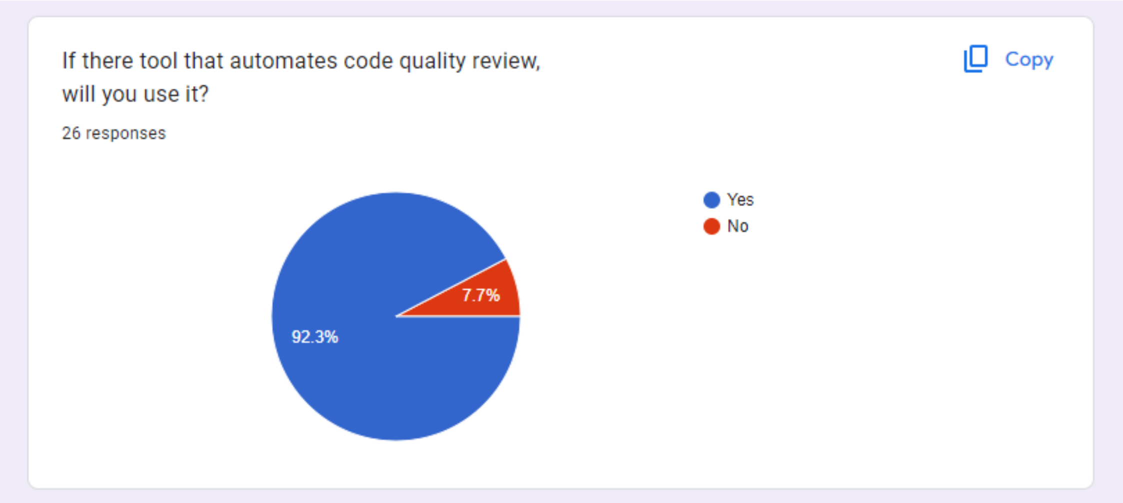
How important is real-time feedback in the code review process for you?

25 responses

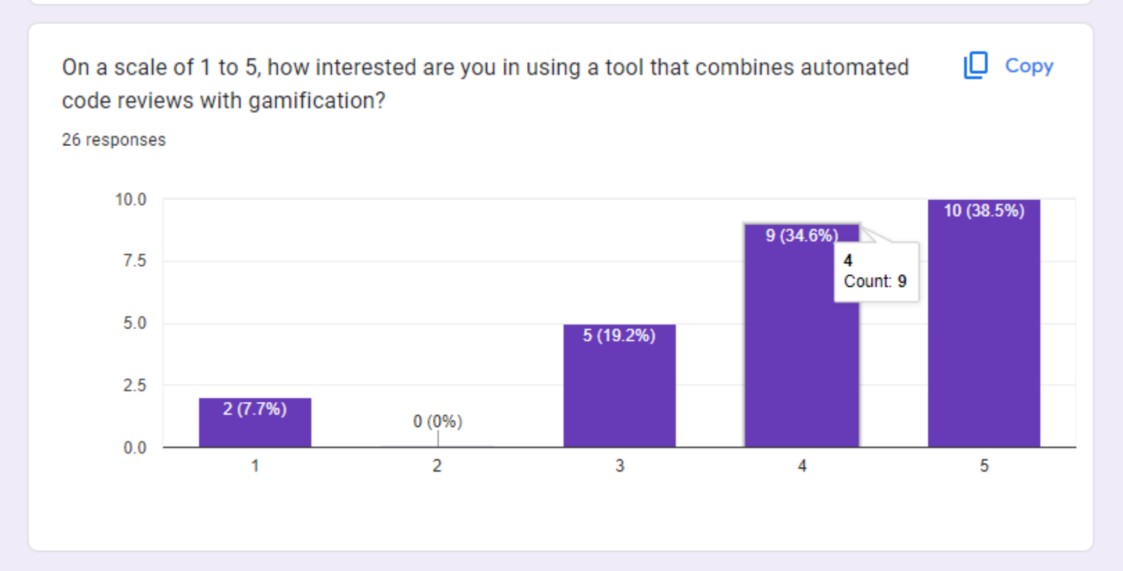


what is most important quality attribute for you ?

26 responses


If there tool that automates code quality review, will you use it?

26 responses


On a scale of 1 to 5, how interested are you in using a tool that combines automated code reviews with gamification?

26 responses

12 Reference

References

[1] Jagriti Sikka, Kushal Satya, Yaman Kumar, et al. “Learning based Methods for Code Runtime Complexity Prediction”. In: *Learning based Methods for Code Runtime Complexity Prediction*. Vol. 1. 6. arxiv. 2019, pp. 1–14.

[2] Mingi Jeon, Seung-yeop Baik, Joonghyuk Hahn, et al. “Deep Learning-based Source Code Complexity Prediction”. In: *Deep Learning-based Source Code Complexity Prediction*. Vol. 1. 6. ICLR. 2023, pp. 1–22.

[3] Srdan Popic, Gordana Velikic, Hlavac Jaroslav, et al. “The Benefits of the Coding Standards Enforcement and it’s Influence on the Developers’ Coding Behaviour: A Case Study on Two Small Projects”. In: *The Benefits of the Coding Standards Enforcement and it’s Influence on the Developers’ Coding Behaviour: A Case Study on Two Small Projects*. Vol. 28. 6. Telecommunications Forum (TELFOR). 2018, pp. 1–4.

[4] Jürgen Börstler, Kwabena E. Bennin, Sara Hooshangi, et al. “Developers talking about code quality”. In: *Developers talking about code quality*. Vol. 28. 6. Empirical Software Engineering. 2023, pp. 1–31.

[5] Mutiat A. Ogunrinde and Solomon O. Akinola. “Performance Evaluation of a Code Complexity Measurement Tool: An Empirical Approach”. In: *Performance Evaluation of a Code Complexity Measurement Tool: An Empirical Approach*. Vol. 13. 1. AFRICAN JOURNAL OF COMPUTING and ICT. 2020, pp. 55–69.

[6] Gao Hao, Haytham Hijazi, João Durães, et al. “On the accuracy of code complexity metrics: A neuroscience-based guideline for improvement”. In: *On the accuracy of code complexity metrics: A neuroscience-based guideline for improvement*. Vol. 16. 1. Frontiers in Neuroscience. 2023, pp. 1–24.

[7] Maja Videnovik, Elena Vlahu-Gjorgievska, and Vladimir Trajkovik. “To code or not to code: Introducing coding in primary schools”. In: *To code or not to code: Introducing coding in primary schools*. Vol. 29. 1. Computer Applications in Engineering Education. 2020, pp. 1132–1145.

[8] Alexander Shypula, Aman Madaan, Yimeng Zeng, et al. “LEARNING PERFORMANCE-IMPROVING CODE EDITS”. In: *LEARNING PERFORMANCE-IMPROVING CODE EDITS*. 2023, pp. 1–23.

[9] Radosław Roszczyk, Wdowiak Id, Kamil Rybiński, et al. “BalticLSC: A low-code HPC platform for small and medium research teams”. In: *BalticLSC: A low-code HPC platform for small and medium research teams*. IEEE. 2021, pp. 1–4.

[10] Jan Bielecki and Michał Miałek. “Estimation of execution time for computing tasks”. In: *Estimation of execution time for computing tasks*. Cluster Computing. 2022, pp. 3944–3956.

[11] Vard Antinyan, Miroslaw Staron, and Anna Sandberg. “Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time”. In: *Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time*. Vol. 22. 1. Empirical Software Engineering. 2017, pp. 3057–3087.