Mining and Fusing Productivity Metrics with Code Quality Information at Scale

Harsh Mukeshkumar Shah, Qurram Zaheer Syed, Bharatwaaj Shankaranarayanan, Indranil Palit, Arshdeep Singh, Kavya Raval, Kishan Savaliya, Tushar Sharma
Dalhousie University
Halifax, Canada
{harsh.shah, qurram.syed, bharatwaaj, indranil.palit, singh.arsh, kv286760, ks317715, tushar}@dal.ca

***Abstract*—Productivity in software development is a complex, multi-faceted concept expressed as a combination of *effectiveness* and *efficiency*. From a quantitative lens, productivity is often interpreted from a collection of activities and metrics such as the number of commits, lines of code added and removed, and the number of issues closed. Software development team managers often seek to track developers' activity and productivity for short-term planning and medium-term team performance measurement. Existing tools and platforms analyze and visualize individual aspects of developers' activity, productivity, or quality. However, a tool that fuses multiple information streams representing productivity and quality aspects is missing. The proposed tool QConnect fills the gap by mining, analyzing, and fusing information from software development-relevant streams. QConnect, on the one hand, mines the repository and issue tracking metadata from GitHub and Jira issue tracking system; on the other hand, it gathers information related to code quality using external tools *Designite* and *RefactoringMiner*. By tying-in productivity measures with code quality information, stakeholders can assess not only *how fast* but also *how well* the project is progressing.**
***Demo:* tool website and demo video.**

***Index Terms*—Developer activity, repository mining, productivity, software quality, dashboard, software project management.**

I. Introduction

Productivity in software engineering is far from a simple ratio between output and input; it is a complex multi-faceted concept expressed as the combination of *effectiveness* and *efficiency* [1]. When it comes to concrete metrics, software engineering community no longer considers Lines of Code (LOC) alone as a good metric for software productivity [2], [3]. Jaspan *et al.* [4] argues that there is no single metric that can be used universally to measure productivity. Despite that, software project managers often seek to track developers' activity and productivity [4]. Tracking teams' activities and productivity is used for short-term planning and measuring performance of the team [5], [6]. Software development organizations use metrics such as LOC, function points, and velocity [1]. In addition, such organizations seek other data points (such as repository metrics including open and closed issues) as well as code quality and quality assurance measures to gain further insights about their development effort.

There has been some attempts from academic researchers and commercial tool vendors to capture project development activities and metrics of productivity. Axosoft [7] offers a dashboard to track progress for an agile project. Jira dashboard from Atlassian tool suite [8] also allows tracking progress based on Jira tickets. Similarly, Bitergia's analytics platform [9] and DueCode [10] generate reports covering various aspects of productivity and contributions. On the other hand, there has been various tools and platforms such as SonarQube [11], CodeScene [12], and *Designite* [13] to analyze source code and provide code quality information. However, the present set of tools target either *effectiveness* or *efficiency* aspect of a software development effort. In other words, the present set of tools offer, with varying degree of support, either project activity and productivity metrics or code quality reports; they do not provide a quality-aware development status where software code quality information is presented in the context of project activities and productivity measures. Due to this, a project manager can observe only a partial picture of the ongoing effort in a software development project.

QConnect aims to fill the gap by offering a quality-aware dashboard for developers' activity and productivity. The *goal of the tool is to mine information from multiple relevant sources within the software development domain and fuse them together to reveal insights that are visible only when multiple information streams are fused.* The tool offers a comprehensive set of development activity and productivity metrics such as commit frequency for all repositories, code churn (by team, user, and by file), pull-request frequency and time to close pull-requests, commits against Jira tickets, and Jira tickets closing frequency. In addition, QConnect provides detailed information about the evolving code quality with development activities. The information includes detected smells at architecture, design, implementation, and test granularity over time, smell density by smell type over time, and refactoring frequency (by team and by commit) along with specific refactoring types. Tying-in productivity measures with code quality information from multiple sources, a project manager as well as the entire development team can assess not only *how fast* but also *how well* the project is progressing. Furthermore, QConnect can also be used to infer data-driven insights about potential bottlenecks and issues.

II. Related work

There have been many attempts to build dashboards targeting team-level and personal developer workspace. RescueTime [14] allows developers to set personal software development goals and provides a productivity score. WakaTime [15] integrates with IDEs and provides a dashboard with metrics such as programming language used. Codealike [16] also integrates itself with Visual Studio IDE and offers features for developers to retain their focus. QScored [17] analyzes GitHub open-source repositories and offers code quality metrics and code smells in the form of a dashboard.

A set of tools and platforms offers a dashboard for software project activities and productivity. GitHub provides a basic set of plots such as commit and contribution frequency to visualize project activity. One may use Atlassian tool suite [8] to track progress based on their Jira dashboard. Similarly, Axosoft [7] offers a sprint dashboard to track progress for an agile project. Finally, Bitergia [9] developed an analytics platform to compute project metrics and to generate reports covering various aspects of productivity and contributions.

A set of tools focus on quality analysis such as CodeScene [12] and Code Climate [18]; these tools integrate with GitHub directly and are also available via the GitHub Marketplace. CodeScene offers a behavioral quality analysis of the tool to keep the code maintainable. Apart from them, there are variety of tools that work independently *i.e.,* without integration with IDEs or code hosting platforms, but provide dashboard containing comprehensive code quality information. These tools include SonarQube [11] and *Designite* [13].

Insights lie in combining relevant information streams within software development. To the best of our knowledge, the present set of tools provide standalone pieces of information and do not combine the information originating from multiple tools, processes, and artifacts within a development context. Therefore, the existing tools do not offer insights that can be seen only by combining the information. QConnect aims to cover the gap by fusing detailed productivity metrics with code quality information.

III. The proposed tool—QConnect

A. Architecture

Figure 1 presents an overview of the tool's architecture. The implementation is divided into two major modules—front-end and back-end. The front-end is developed using *React*; the back-end is written in Python using *Django* framework. The front-end provides a set of interactive visualizations categorized into three dashboards — *summary, productivity, and quality*. The tool offers a set of filters (*i.e.,* repository, time-frame, and user) to customize the view wherever applicable.

In the back-end, repository processors carry out the heavy-weight functionality. *Bootstrap repository processor* takes one repository and fetches all the required information from GitHub and Jira with the help of back-end services when the repository is freshly added to the tool. *Periodic repository processor* fetches the needed information since the last time

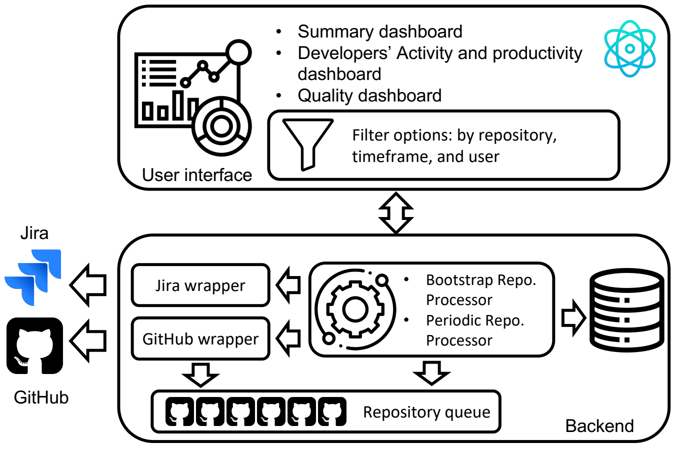

Fig. 1. Overview of the QConnect architecture

it was analyzed. The repository processors rely on micro-services implementing GitHub and Jira wrappers to extract the required information from these platforms. GitHub wrapper uses GitHub APIs to get metadata about branches and pull requests. Jira SDK APIs are used to get information about epics, assignee, issues, and sprints. Each incoming request to analyze a GitHub repository is first put on a *repository queue*. The queue helps the tool manage its workload and analyze the repositories one by one based on the available hardware infrastructure. All the obtained information is stored in a *MongoDB* database. The repository processors use external tools to gather code quality information. For example, *Designite* and *DesigniteJava* [19] generate comprehensive code quality information (in the form of smells and metrics) for C# and Java projects respectively. The processors also use *RefactoringMiner* [20] to identify the carried out refactorings in each commit.

The back-end of the tool is realized using multiple micro-services. Figure 2 presents the workflow at the services level. The *Bootstrap repository processor* in Figure 1 is realized by the *workflow manager* service. When a repository is added for analysis, the main application *i.e.,* QConnect service invokes workflow manager service that in turn calls knows the steps and sequence of the repository mining. For majority of Version Control System (VCS) operations are performed locally by using PyDriller [21]. For the rest of the information (such as pull-request metadata), the application uses GitHub APIs.

B. Features

We summarize below the features and functionality supported by QConnect.

a. Summarized view of developers' activity and quality: The summary dashboard shows key metrics and metadata of all the repositories that the logged-in user have. These metrics and metadata includes *active contributors*, *total number of commits*, *total code churn* (number of LOC added and removed) as well as *smell density* across all the repositories in the selected time-frame. The tool also presents a set of graphical visualizations summarizing the development activity and

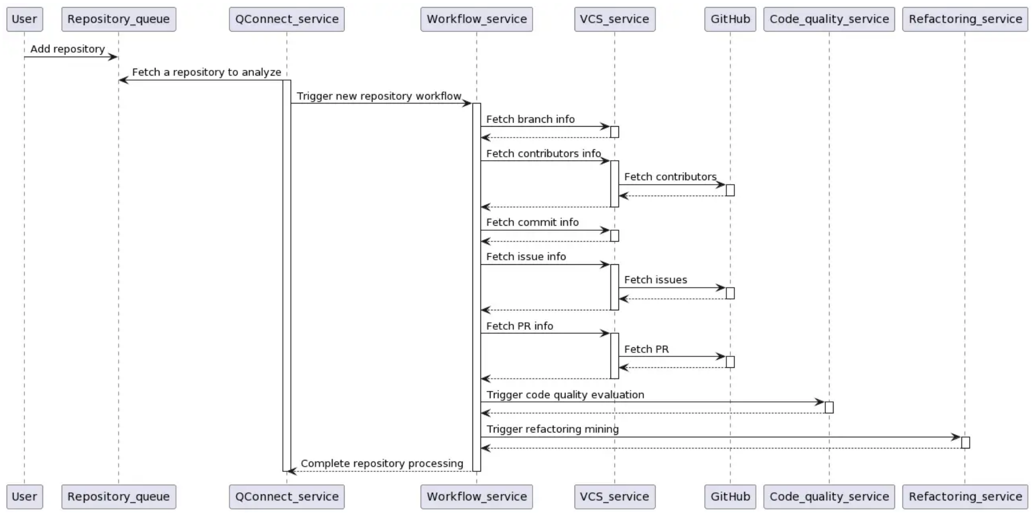

Fig. 2. Adding a repository workflow among QConnect micro-services

productivity using various measures. The tool's visualizations include the *total number of refactorings and smell counts* by day, and *lines of code added or deleted* over time during the selected time-frame. In addition, the tool includes a plot on the *top contributors* in terms of the total amount of commits made across all repositories. By-default, time dimension of all the plots take two weeks as the duration; however, the tool allows the user to change the time-frame.

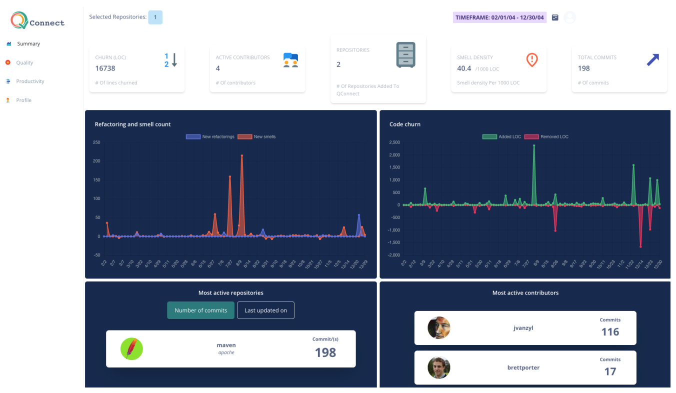

Fig. 3. Summary dashboard of QConnect

b. Developers' activity and productivity dashboard: The productivity dashboard comprises a number of plots to visualize various metrics associated with developers' activity and productivity. To show quantified contributions, the tool includes a plot showing *total contributions* in terms of total number of commits by users. Another plot shows active branches by number of commits. Furthermore, the tool also provide a chart showing *total* and *active* contributors in the selected time-frame. The dashboard also includes a plot showing active and new pull requests in a repository.

Many software development organizations maintain a separate Jira board as their issue tracking system while maintaining their repository on GitHub. The tool integrates Jira issue tracking system and produces plots to generate insights by consolidating information from Jira board and GitHub repository. For example, the dashboard provides the most active set of issues by number of commits by the team. Such a plot reveals the effort spent by the team on specific issues. Similarly, the tool offers a finer-grained plot showing average time taken to close issues by users. Furthermore, the dashboard also provides plots on average time to close issues, and a ratio of closed and open issues in a given time-frame.

c. Code churn and code quality dashboard: The dashboard includes code churn in terms of lines of code added or removed in the selected time-frame. In addition, a treemap shows the changed files where the size of churn reflects in the size of the rectangle for the file. The tool uses *DesigniteJava* [19] to analyze Java repositories and to produce a comprehensive code quality report including architecture, design, implementation, testability, and test smells along with code quality metrics. The dashboard uses the code quality information and fuse it with repository metrics. The dashboard includes a plot showing evolution of smell density by the granularity (*i.e.,* architecture, design, and implementation) and scope (*i.e.,* testability and test) of the detected smells. Another plot shows added smells by specific smell type. Furthermore, QConnect integrates *RefactoringMiner* tool [22] to identify the performed refactorings per commit. The dashboard shows a plot exhibiting the total number of specific refactoring techniques applied in a given time-frame. These plots provide a comprehensive understanding of code quality in the context of changes made in the source code.

IV. Preliminary Evaluation and Case Study

We carried out a preliminary evaluation of the developed tool. As a subject system, we required an open-source software with significantly long commit history that is written either in Java. We also needed access to Jira issue tracking board of the subject system. We selected Apache Maven[1] as our subject system; at the time of analyzing the project, the project had more than 11.8 thousand commits starting from the year 2003. The project is written in Java programming language and the Jira issue tracking can be accessed online[2].

Property	**Value**
# of analyzed commits	11,829
Lines of code analyzed	403,237,410
# of Jira issues	6,288
# of PRs fetched	712

We analyzed all the commits of the repository in the main branch using *DesigniteJava* and *RefactoringMiner*. We obtained commit metadata (such as commit author, date, and code churn), and fetched pull-request and Jira issue metadata from GitHub and Jira respectively. We randomly selected ten commits and manually verified the extracted or computed data from GitHub, Jira, *DesigniteJava*, and *RefactoringMiner*. In the rest of the section, we take four specific use-cases in the context of the selected subject system to discuss the applicability of the tool.

Use-case 1: *How a stakeholder can learn about the progress of the project in a quick glance?* — The summary dashboard provides a consolidated summarized view of all the repositories analyzed with the tool for a user. Figure 3 shows the key summary metrics such as *code churn*, *smell density*, and *total commits* in the selected time-frame. *Refactoring and smell count* plot shows the total number of new smells and refactorings. For example, the figure shows significant number of smells introduced in the project during the Jul-Aug 2004 period. At the same time, we can observe that the code churn plot also shows high activity in the same period.

Use-case 2: *How productive the team was during the selected period?* — Various measures and plots exhibit different aspects of team productivity. First, one can learn about the *top contributors* of the team in terms of number of commits from the summary dashboard. The *number of active and total contributors* can be learned from productivity dashboard. Figure 4 shows that the average rate of closing issues fluctuating in the selected period. Similarly, the *average rate of closing pull-requests* touched 80 days during the selected time-frame. *Resolved and new pull request (PR) plot* exhibits the amount of pending work in the repository. The figure shows the number of active PRs reached up to 90 but subsequently came down to zero in the selected duration.

Use-case 3: *What is the effect of code churn on code smells and refactoring?* — *Code churn plot* along with *smell density and detailed smell count* plots as well as with *refactoring plot* can help us understand the code changes and their effects. The *code churn* plot (not included in the text due to space) shows a contributor made significant changes on Mar 19. The *refactoring plot* in the same figure shows that the changes

[1]https://github.com/apache/maven

[2]https://issues.apache.org/jira/projects/MNG/issues/MNG-7316

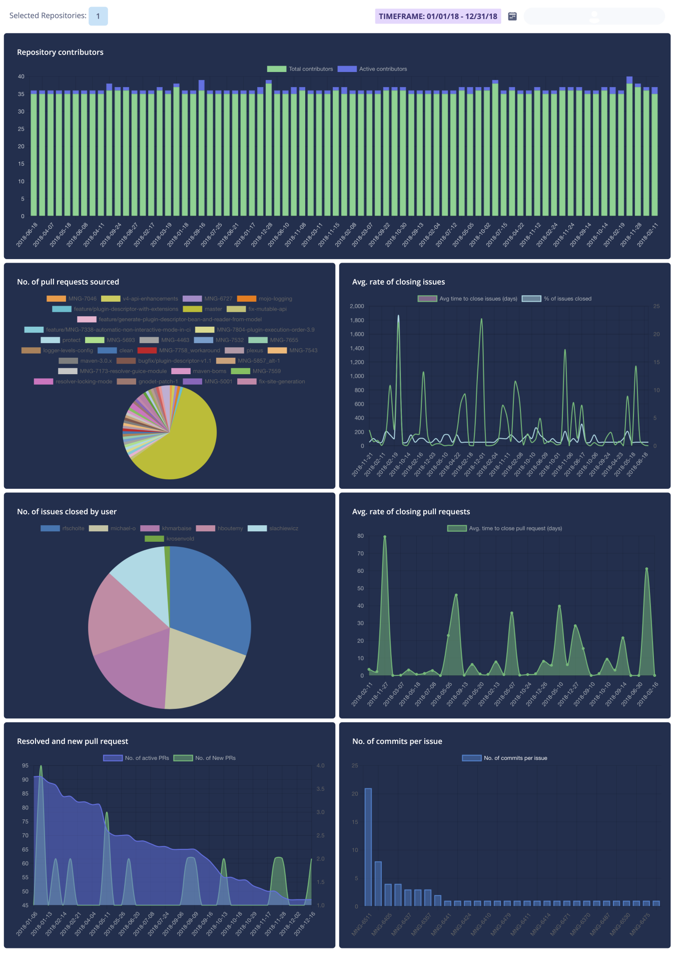
Selected Repositories: 1
TIMEFRAME: 01/01/18 - 12/31/18
Repository contributors
Total contributors
Active contributors
No. of pull requests sourced
Avg. rate of closing issues
Avg time to close issues (days)
% of issues closed
No. of issues closed by user
Avg. rate of closing pull requests
Avg. time to close pull request (days)
Resolved and new pull request
No. of active PRs
No. of New PRs
No. of commits per issue

Fig. 4. Developers' activity and productivity dashboard of QConnect

introduced eight refactorings. Similarly, a rise in smell density on Mar 23 is visible both in *smell density* and *smells details* plots. Furthermore, the changed files and their corresponding change size can be observed in *churn per file* plot. This plot includes data from three sources *viz.* version control system *i.e.,* GitHub, RefactoringMiner, and DesigniteJava.

Use-case 4: *Which issues are consuming significant effort in a given duration?* — One of the most insightful plots is the *number of commits per issue* that shows the specific issues against which the contributors are putting their efforts. The figure reveals that most of issues received one commit each to solve them; one of the issues required up to 22 commits. The plot pool-in data from issue-tracking system (Jira in our case) and version control system.

V. Challenges and Limitations

We faced various engineering challenges. For example, despite we collect the majority of the VCS metadata locally, the tool depends on GitHub APIs to collect metadata about pull-requests and contributors. It is challenging to ensure metadata collection due to the maximum limit (currently $5,000$ calls per hour) on the API calls imposed by GitHub. The challenge

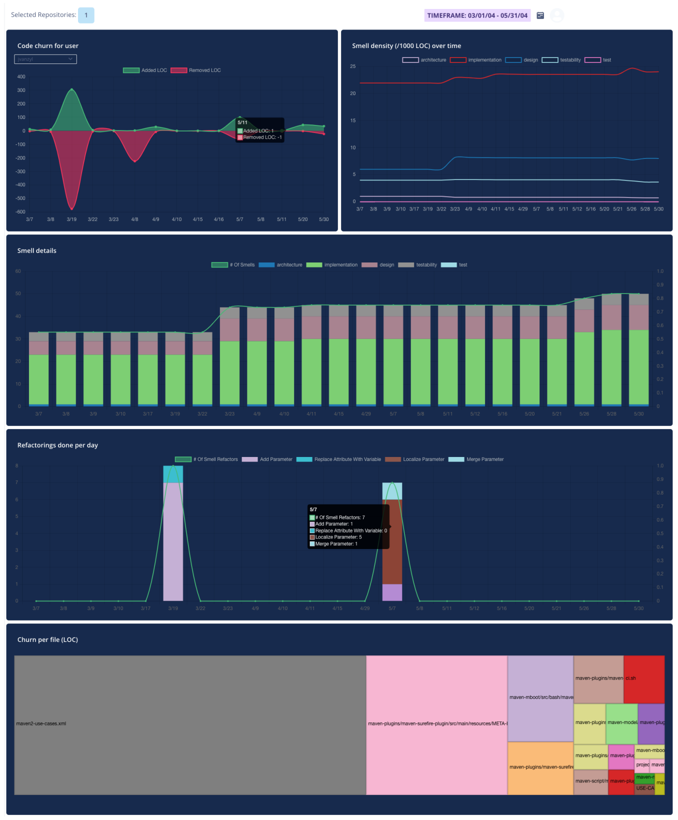

Fig. 5. Code quality dashboard of QConnect

was to ensure not only to restart the metadata collection once we regain our API limit but also to fairly allocate the resources to all the repositories in the queue. To mitigate the challenge, we implement a resource handler for the API that keeps track of the current limit, pauses the API calls to GITHUB when the application is nearing limit exhaustion, restarts the metadata collection when the limit replenishes, and allocates the metadata collection resources to other repositories in round-robin fashion.

A limitation of the tool is the availability of limited hardware resources. We intend to analyze each commit with code quality analysis tools that takes considerable hardware resources. As the number of repositories to be analyzed increases, the need of scalable hardware infrastructure also rise. We aim to extend the tool as a native cloud application so that the tool can spawn the required number of attainable virtual machines to analyze the new analysis requests. Another limitation of the tool is the supported languages for code quality analysis. Currently, we use *Designite* tools for C# and Java languages; however, tool support to generate a comprehensive code quality report for other languages is presently lacking.

VI. Conclusions and Future Work

QConnect bridges developer activity and productivity metrics with code quality analysis. The tool mines, combines, and presents the information from multiple sources to provide a consolidated and comprehensive perspective of software development progress. The tool enables a software development team and its management to ensure that the team is progressing *w.r.t.* their business goals while keeping a focus on technical excellence. The visualization aids offered by the tool helps the team understand fine-grained analysis of the activities that could be used for performance evaluation and bottleneck identification. We identify many opportunities to improve. For example, we would like to make the tool a truly scalable cloud-native application to handle the varying hardware resource requirements. We are working to improve the user interface and user experience. We aim to extend the platform to cover other code hosting platform apart from currently supported GITHUB, such as GitLab. We also would like to explore the possibility of integrating additional external tools to enrich the information with the focus on integrating the data that tells a cohesive story.

References

[1] S. Wagner and F. Deissenboeck, *Defining Productivity in Software Engineering*. Berkeley, CA: Apress, 2019, pp. 29–38.
[2] T. C. Jones, “Measuring programming quality and productivity,” *IBM Systems Journal*, vol. 17, no. 1, pp. 39–63, 1978.
[3] Boehm, “Improving software productivity,” *Computer*, vol. 20, no. 9, pp. 43–57, 1987.
[4] C. Jaspan and C. Sadowski, *No Single Metric Captures Productivity*. Berkeley, CA: Apress, 2019, pp. 13–20.
[5] T. Kanij, J. Grundy, and R. Merkel, “Performance appraisal of software testers,” *Information and Software Technology*, vol. 56, no. 5, pp. 495–505, 2014, performance in Software Development.
[6] R. M. Parizi, P. Spoletini, and A. Singh, “Measuring team members’ contributions in software engineering projects using git-driven technology,” in *IEEE Frontiers in Education Conference (FIE)*, 2018, pp. 1–5.
[7] “Axosoft,” https://www.axosoft.com/, online; accessed May 26, 2023.
[8] “Atlassian tool suite,” https://www.atlassian.com/blog/agile/jira-software-agile-dashboard, online; accessed May 26, 2023.
[9] “Bitergia’s analytics platform,” https://bitergia.com/about/, online; accessed May 26, 2023.
[10] DueCode, “DueCode,” https://duecode.io/, online; accessed May 26, 2023.
[11] “SonarQube,” https://www.sonarqube.org/, online; accessed May 26, 2023.
[12] A. Tornhill, “CodeScene,” https://codescene.com, online; accessed May 26, 2023.
[13] T. Sharma, “Designitejava (enterprise),” Sep. 2019, http://www.designite-tools.com/designitejava. [Online]. Available: https://doi.org/10.5281/zenodo.3401802
[14] “RescueTime,” https://www.rescuetime.com/, online; accessed May 26, 2023.
[15] “WakaTime,” https://wakatime.com/, online; accessed May 26, 2023.
[16] “Codealike,” https://marketplace.visualstudio.com/items?itemName=Codealike.Codealike, online; accessed May 26, 2023.
[17] V. Thakur, M. Kessentini, and T. Sharma, “QScored: An Open Platform for Code Quality Ranking and Visualization,” in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 818–821.
[18] B. Helmkamp, “Code Climate,” https://codeclimate.com, online; accessed May 26, 2023.
[19] T. Sharma, “DesigniteJava,” Dec. 2018, https://github.com/tushartushar/DesigniteJava. [Online]. Available: https://doi.org/10.5281/zenodo.2566861
[20] N. Tsantalis, M. Mansouri, L. M. Eshkevari, D. Mazinanian, and D. Dig, “Accurate and efficient refactoring detection in commit history,” in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE ’18, 2018, pp. 483–494.
[21] D. Spadini, M. Aniche, and A. Bacchelli, *PyDriller: Python Framework for Mining Software Repositories*, 2018.
[22] N. Tsantalis, A. Ketkar, and D. Dig, “Refactoringminer 2.0,” *IEEE Transactions on Software Engineering*, 2020.