QScored: An Open Platform for Code Quality Ranking and Visualization

Vishvajeet Thakur
Himachal Pradesh University
Shimla, India
vishvajeet10gfs@gmail.com

Marouane Kessentini
University of Michigan
Dearborn, USA
marouane@umich.edu

Tushar Sharma
Siemens Corporate Technology
Charlotte, USA
tusharsharma@ieee.org

***Abstract*—Though abundant source code repositories are available on code repository hosting platforms, their detailed code quality information is not available readily. Software engineering researchers often need to select a set of high-quality repositories. Despite the code quality is an important concern for repository selection, the lack of this information makes researchers depend on alternatives such as the number of issues and the number of stars associated with repositories. Furthermore, practitioners expect user-friendly visual ways to assess the quality of their projects during the evolution of the codebase without putting a considerable effort. We propose an open platform *QScored* to fill the gap for both researchers and practitioners. The platform hosts detailed code quality analysis information for a large number of repositories (currently more than eleven thousand containing more than 171 million LOC), computes quality score and assigns relative ranking of the hosted repositories based on detected architecture, design, and implementation smells, as well as offers a comprehensive set of visualization aids for code quality aspects. Furthermore, the platform provides REST APIs to search repositories based on their code quality scores and ranking of hosted software projects.**
Video of the demo: https://youtu.be/-IgvjGV-2X0

***Index Terms*—Code quality, visualization, quality score, quality ranking, quality badge.**

I. Introduction

Software development is a long, complex, and continuous process. High code quality makes software maintainable [1], [2] and in turn, helps to keep the productivity of a software system high [3]. Code smells [1] and code quality metrics [4] are the common mechanisms used traditionally to identify issues that impact the code quality of a software system. The software engineering community has explored not only numerous tools and techniques to detect code smells but also various aspects of code smells including their types, causes, and impacts [5].

Today, abundant source code repositories are available on code repository hosting platforms such as GitHub. However, a detailed code quality analysis information is not available for the hosted repositories unless we use an existing code quality analysis tool to analyze the desired repository ourselves. Furthermore, even if one analyzes a set of repositories, a relative scale of code quality is non-trivial to establish. For empirical research, it is often the case that the software engineering researchers have to select a few high-quality repositories. Due to the lack of a standard ranking mechanism based on code quality of a large number of repositories, researchers adopt alternative mechanisms such as the number of issues and the number of stars for repository selection. Though there have been some attempts to address this issue, for instance RepoReapers [6] does not consider detailed code quality measures for their repository evaluation.

Visualizing various aspects of software code quality provides a structure to the analysis and helps comprehending the information generated by the code quality analysis tools. Though, existing tools provide visualization aids, their coverage to different code quality aspects as well as visualizing mechanisms differ significantly. Therefore, *ready made* visual aids in a common template, applicable for different programming languages, for visualizing key code quality aspects such as code smells at different granularities, code quality metrics, and dependency diagram could offer great help in understanding the code quality of open-source projects; existing tools and platforms do not support the above need.

In this paper, we introduce *QScored*[1]—an open platform for code quality ranking and visualization. The key contributions of the platform are listed below.

- **An open platform:** QScored is an open and free platform that not only provides code quality information for a large number of open-source repositories (more than $11,300$ at the time of writing this paper and increasing) analyzed from existing tools but also allows its users to upload their code quality analysis report of their projects to the platform to determine a quality score and a relative quality ranking.
- **Quality ranking:** QScored computes a quality score and a relative quality rank for all of its hosted projects based on the detected architecture, design, and implementation smells.
- **Language-agnostic code quality visualization:** QScored offers a detailed code quality visualization that includes a dependency graph among components (namespaces in C# or packages in Java), smells distribution among projects' components by their types and their location, and code quality metrics view for the entire project. Furthermore, the platform publishes the XML template that provides a common *information exchange format* so that the analysis

[1]http://www.qscored.com

report of other tools, apart from that are used by QScored presently, complying to the template can also be uploaded to the platform.

- **REST APIs:** The platform offers a set of REST APIs to upload code analysis reports as well as to search a set of projects based on various parameters.
- **Quality badges:** Software development teams may utilize the quality badges indicating quality score and rank of their project and put it within their repository web-page.

II. Related Work

Though abundant repositories are hosted on websites such as GitHub, a large number of repositories among them are either too small, inactive, or low-quality [7]. To separate them from high-quality active repositories, the software engineering community has proposed a few mechanisms. GHTorrent [8] provides a method to query GitHub projects, obtain metadata, and select projects based on custom criteria. RepoRepears [6] analyzed a large number of GitHub repositories and evaluated them based on eight dimensions (architecture, community, continuous integration, documentation, history, issues, license, and unit testing).

The software engineering community has proposed many tools and techniques to identify issues that affect code quality [5]. Apart from identifying metric violations and code smells, researchers have proposed methods to visualize the analyzed information [9]–[11]. For example, E-Quality [12] is a software quality visualization tool that automatically extracts quality metrics and class relations from Java source code and visualizes them on a graph-based interactive visual environment. Specifically for code smells visualization, researchers have attempted techniques to visualize key aspects of smells. For instance, AlTarawneh et al. [13] presents sunburst diagrams to show not only the code smells frequency but also to present their location. Ambient view developed by Murphy-Hill et al. [14] shows code smells within IDE in the form of *petals*. Similarly, Parnin et al. [15] presents various techniques to visualize code smells covering key aspects of the smells. Apart from academic attempts, presently used tools within the developer community, such as NDepend[2], SonarQube[3], and Designite [16], also use various visualization aids to present smells and metric violations.

All of the above-mentioned tools and technique for code smell detection and visualization could offer their services only when a user analyzes the desired set of repositories by herself that she would like to evaluate. By developing QScored, we attempt to remove the manual step for open-source repositories. With code quality aspects already available to a researcher, the effort to investigate the projects of interest can be reduced significantly. Apart from ready-made quality information availability, the platform offers common visualization for all analyzed repositories even if they belong to different programming languages as well as provides means to search open-source repositories based on code quality criteria.

[2]https://www.ndepend.com/

[3]https://www.sonarqube.org/

III. QScored: Mechanism

A. Overview

Figure 1 presents an overview of the QScored functionality. QScored allows its users to upload the analyzed code quality analysis report generated from Designite [16] or any compatible tool. A code analysis tool is compatible with QScored if the tool produces code quality analysis report in the predefined XML format published by QScored. In addition, QScored deploys a program referred to as *QScored agent* that keeps downloading open-source repositories from GitHub, analyzing them using either Designite [16] or DesigniteJava [17], and uploads the analyzed information to QScored using REST APIs offered by the platform. When a new project (or a new version of an existing project) is uploaded, QScored calculates its quality score. Periodically (currently every week), QScored computes quality ranking for all the projects in its corpus. Quality score is the weighted sum of smell densities and hence the lower the quality score, the better the project is from code quality perspective.

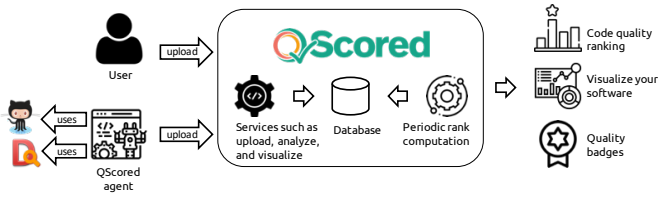

Fig. 1. Overview of the platform functionality

Table I presents some key size metrics for the developed platform at the time of writing this paper. QScored uses PostgreSQL as its database which is hosted on a scalable cloud-based storage. We measured time to analyze and upload 100 randomly chosen repositories; it takes on an average 21.8 seconds to analyze and 4.05 seconds to upload (including validation of uploaded file and storing it in the database) per repository.

Entity	Java	C#	Entity	Java	C#
LOC	93,830,016	77,475,461	#Repositories	6,360	4,970
#Types	1,139,657	764,496	#Components	130,492	98,491
#DS	913,578	792,550	#AS	116,803	65,859
#IS	5,852,030	3,207,806			

TABLE I
KEY SIZE METRICS FOR THE CORPUS OF THE PLATFORM. AS, DS, AND IS REFER TO ARCHITECTURE, DESIGN, AND IMPLEMENTATION SMELLS RESPECTIVELY

B. Generating Code Quality Analysis Report

We use Designite [16] and DesigniteJava [17] for C# and Java repositories respectively to analyze source code and obtain a detailed code analysis report. Each such report contains detected architecture, design, and implementation smells as well as commonly used class and method metrics. Designite detects seven kinds of architecture, 20 kinds of design, and eleven types of implementation smells. Sharma et al. [18] provide a detailed description of smells as well as the tool validation for Designite. Specific commands to analyze a

repository and generate code quality analysis report in XML format can be found in online documentation.[4]

C. Quality Score and Rank

QScored computes a quality score for each project in its corpus based on the detected smells. Quality rank is computed periodically (currently, weekly) based on the quality score of the projects; a lower number of quality score is desired and hence project with the smallest quality score is assigned the highest quality rank. The quality score is computed as follows.

$$\text{Quality score} = \frac{w_a \times ASD + w_d \times DSD + w_i \times ISD}{w_a + w_d + w_i} \quad (1)$$

Here, ASD, DSD, and ISD refer to architecture, design, and implementation smell density respectively. Smell density [18] is a normalized metric representing the total number of smells in each one thousand lines of code; it enables the comparison of projects differing in size. w_a, w_d, and w_i refer to weight assigned to architecture, design, and implementation smell density respectively.

1) Developers' survey: We carried out a survey targeting experienced software developers to obtain suitable weights for architecture, design, and implementation smells in quality score computation. We designed a short questionnaire containing two questions. The first question asked the number of years of software development experience. The second question presented the following scenario to the participants: *"Imagine you are working on a large software project which suffers from many code quality issues at architecture, design, and implementation granularities. Given* 100 *units of time available to you for addressing these issues, in what proportion you would dedicate your time towards issues correspond to each granularity considering their impact on the overall code quality of the project?"* We provided three sliders for each granularity that may take a value between 0 to 100 with step-size 5.

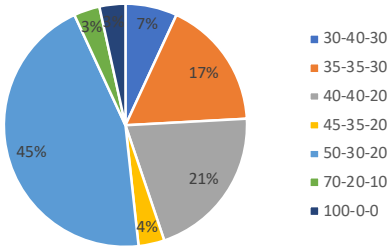

Fig. 2. Responses from the survey showing weights for architecture, design, and implementation issues in x-y-z format

We sent this survey to 51 software developers and architects across our network. We received total 31 responses; however, two of the responses were not valid (sum of the weights exceeded 100) and hence we discarded them. The participants shown high experience level; 7% belonged to $1-2$ years, 28% to $3-5$ years, 31% to $6-10$ years, 21% to $11-20$ years, and 14% participants belonged to greater than 20 years experience band. In the response for the second question (shown in Figure 2), a majority of the participants (45%) responded that they would distribute their 50% time on architectural quality issues, 30% time on design quality issues, and 20% time on implementation level code quality issues. Based on this observation, we chose $w_a = 0.5$, $w_d = 0.3$, and $w_i = 0.2$ as weights for equation 1.

D. Visualization

Apart from a consolidated ranking page that offers search and filter existing projects, QScored offers comprehensive visualization aids for code quality aspects for each project. On the summary page, some key figures (such as LOC, number of types, metric violations, and detected smells) are presented as shown in Figure 3. Also, the page shows an interactive dependency graph showing dependencies among components, a pie-chart exhibiting various detected smells, and a treemap showing size and smell density of each component. Detected smells are presented using an interactive sunburst diagram that could also be used to filter the smells based on their type as well as location. Similarly, all the metrics are presented using an interactive pie-chart where the chart shows the code quality of the entire project from the selected metrics perspective.

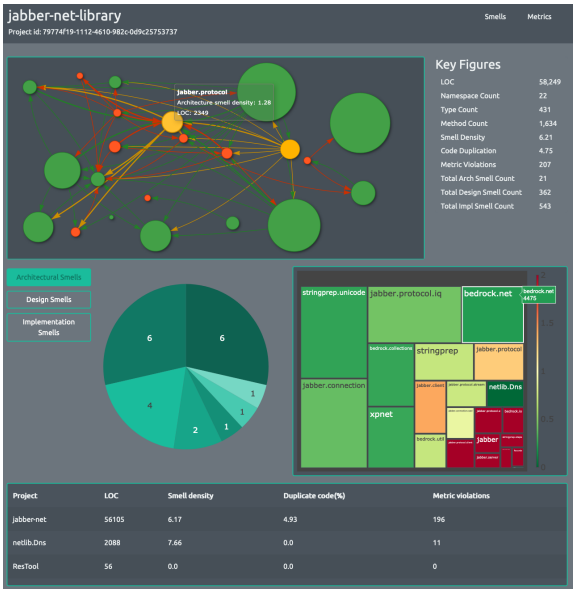

Fig. 3. Screenshot of QScored code quality summary page

Furthermore, the platform publishes the XML template—a common information exchange format so that the analysis report generated from any tool, that comply with the template, can also be consumed by the platform. The XML schema can be found online.[5]

E. REST APIs

QScored supports two APIs—*upload* and *search*. A user has to acquire her API key first from QScored to invoke the APIs.

[4]http://www.qscored.com/docs/upload

[5]http://www.qscored.com/docs/extend_scope

Upload: The `upload` API requires username, project name, version, repository link (optional), is_open_source flag, and API key as inputs along with an XML file containing code analysis report and returns a unique id of the created project if successful. This API can also be used to update an existing project with a fresh analysis on a new version.

Search: The first search API `search_project_by_quality` allows users to search projects based on their quality ranking. One may specify filters *language* and LOC *range* to obtain a list of projects that satisfy the criteria sorted in the order of their quality ranking. The second API `search_project` returns the metadata about the searched projects. The API takes *project name* and *repository link* as inputs. Both of the search APIs returns a list of project metadata containing project name and unique id, repository link, LOC, programming language, as well as quality rank and score. Examples of the above-mentioned APIs can be found online[6].

F. Quality Badges

The platform brings gamification aspects by letting its user display quality rank and score of their projects on their repository web-pages. QScored provides a dedicated link for each project that could be used for the purpose. Any update in the score or rank is reflected in the badge.

G. Privacy Concerns

Privacy is an important concern that we address in the following ways. Users, while uploading code analysis report of their projects, choose whether their project's data and identity will be accessible by others or only by themselves. A project marked as private will be assigned a quality score and rank; however, the identity and the detailed code analysis information will be available only to the owner of the project. Similarly, we implement double indirection to realize badge functionality to hide the users' private API keys from public web-pages or repositories.

IV. Limitations

We identify the following limitations of the platform. First, currently, the platform hosts repositories implemented in either C# or Java. However, we have made the code quality analysis report format open to invite tool builders and researchers to develop their tools and emit output in the desired XML template to enrich the platform. Also, currently, the platform hosts relatively a small number of repositories compared to the publicly available repositories. We plan not only to keep adding new repositories periodically using QScored agent but also re-analyze the changed repositories since the last analysis to build a project-specific quality trend. In this pursuit, we aim to keep our focus on C# and Java repositories that are active (for instance, changed at least once in the last one year).

[6]http://www.qscored.com/docs/api

V. Conclusions

We present an open platform for code quality visualization and quality ranking. The computed quality score and rank considers smells detected at three granularities and provides a convenient mechanism to select repositories based on the code quality. In the future, we would like to extend the platform by offering continuous integration mechanism for software development teams to enable them make the process of analysis and upload the code quality information completely automated.

References

[1] M. Fowler, *Refactoring: Improving the Design of Existing Programs*, 1st ed. Addison-Wesley Professional, 1999.

[2] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, 1st ed. Morgan Kaufmann, 2014.

[3] T. Besker, A. Martini, and J. Bosch, "Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work," *Journal of Systems and Software*, vol. 156, pp. 41 – 61, 2019.

[4] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., 2002.

[5] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, vol. 138, pp. 158 – 173, 2018.

[6] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, Dec 2017.

[7] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, Oct 2016.

[8] G. Gousios, "The ghtorrent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. IEEE Press, 2013, pp. 233–236.

[9] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey," *Journal of Software Maintenance*, vol. 15, no. 2, p. 87–109, Mar. 2003.

[10] P. Caserta and O. Zendra, "Visualization of the static aspects of software: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 913–933, 2011.

[11] S. Bassil and R. K. Keller, "Software visualization tools: survey and analysis," in *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*, 2001, pp. 7–17.

[12] U. Erdemir, U. Tekin, and F. Buzluca, "E-quality: A graph based object oriented software quality visualization tool," in *2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, 2011, pp. 1–8.

[13] R. AlTarawneh and S. R. Humayoun, "Visualizing software structures through enhanced interactive sunburst layout," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 2016, p. 288–289.

[14] E. Murphy-Hill and A. P. Black, "An interactive ambient visualization for code smells," in *Proceedings of the 5th International Symposium on Software Visualization*, 2010, p. 5–14.

[15] C. Parnin, C. Görg, and O. Nnadi, "A catalogue of lightweight visualizations to support code smell inspection," in *4th ACM Symposium on Software Visualization*, 2008, p. 77–86.

[16] T. Sharma, " Designite - A Software Design Quality Assessment Tool," May 2016. [Online]. Available: https://doi.org/10.5281/zenodo.2566832

[17] ——, "Designitejava," Dec. 2018, https://github.com/tushartushar/DesigniteJava. [Online]. Available: https://doi.org/10.5281/zenodo.2566861

[18] T. Sharma, P. Singh, and D. Spinellis, "An empirical investigation on the relationship between design and architecture smells," *to appear in Empirical Software Engineering (EMSE)*, Jun. 2020. [Online]. Available: http://www.tusharma.in/preprints/architecture_smells.pdf