Building a Collaborative Platform for Evaluating and Analyzing Source Code Quality

Tugkan Tuglular
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: tugkantuglular@iyte.edu.tr

Onur Leblebici
Univera
Izmir, Turkiye
email: onur.leblebici@univera.com.tr

Emre Baran Karaca
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: emrekaraca@std.iyte.edu.tr

Naşit Uygun
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: nasituygun@std.iyte.edu.tr

Osman Anıl Hiçyılmaz
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: osmanhicyilmaz@std.iyte.edu.tr

Cem Sakızcı
Research Ecosystems
Izmir, Turkiye
email: sakizcicem@gmail.com

***Abstract* - The typical approach to data analysis is to store, query, and analyze data in a central location. In the case of source code, where multiple organizations or partners in a consortium contribute to a software, the repositories would be distributed and might be private. Within such a setting, one goal would be achieving and maintaining a certain level of source code quality across the consortium. One solution is to consider each partner as a node in a federated network. This paper proposes a federated code quality query and analysis platform. It further presents the features, the design, and the implementation of this platform.**

Keywords - source code quality; federated network; federated query; federated analysis.

I. INTRODUCTION

The proposed method in this paper improves the federated source code quality query and qnalysis platform presented in [1]. There are cases where each partner in a consortium, such as in the NESSI-SOFT project [2] in the Sixth Framework Program and in the MODUS project [3] in the Seventh Framework Program, does not want to share all of its source code but needs to be queried whether holding a pre-determined minimum source code quality level so that a certain level across the consortium is achieved and maintained. For such cases, one solution is to build a federated network so that each node in this network has its privacy, but shares required quality information. This paper considers this setting for source code quality and proposes a Federated Source Code Quality Query and Analysis (FSCQQA) platform. The setting is visualized in Figure 1.

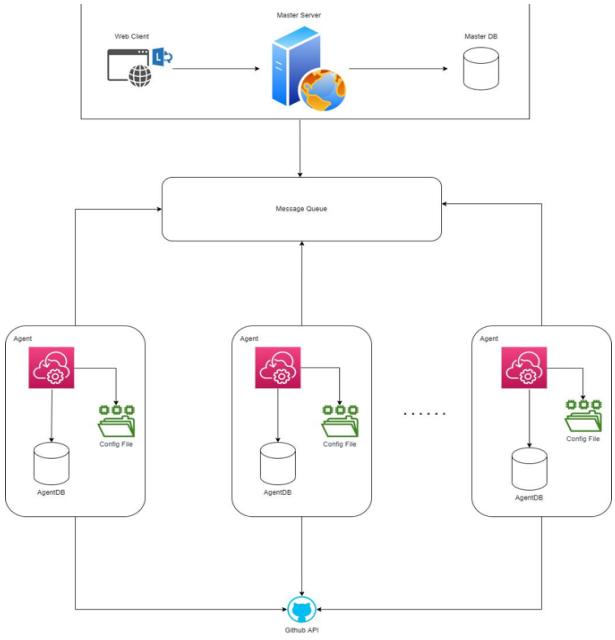

Figure 1. The FSCQQA platform overview.

The FSCQQA platform consists of a central site as seen at the top of Figure 1 and multiple sites, which are peers. It is a kind of peer-to-peer network, where the peers accept and

follow a general policy and corresponding rules. In addition, the central site is responsible for inclusion and removal of peer sites with respect to the general policy. Such platforms are on the rise especially in the health field, where privacy regulations and expectations are high, and accountability is enforced at state level. The proposed FSCQQA platform is one of the early attempts, where the idea is applied to source code, but not health records. Therefore, we believe that there is a practical gain from such a platform proposal.

The proposed platform is not only for consortiums to utilize. A global software company with development sites in various countries can also benefit from the FSCQQA platform. In this setting, concerns like revealing too much information about the software under development and the software development team may be relieved.

The FSCQQA platform offers opportunities for querying and monitoring source code quality across a consortium. This platform can facilitate analyzing how source code improvements are performed and how defect numbers are minimized. The FSCQQA platform has the following features:

- Analyze software quality with defect and source code metrics.
- Share defect and source code metrics with peers and consortium administration/management.
- Follow trends and improve.
- Compile federated historical data on defects and source code quality.

The features are kept at minimum in the paper, but they can be extended easily. To serve these features, the FSCQQA platform provides a data infrastructure, a software stack, and the operations on them. The proposed design is novel. The FSCQQA platform can be used for source code quality and defect prediction in the future.

As of today, there are multi-site software development companies whose sites are globally distributed. Each site is autonomous to some degree, but they are subject to central management rules. In such a setting, tracking each site's software quality and achieving an overall performance is not easy. Such a platform would be beneficial to them as well.

The paper is organized as follows: Section II presents the bug, or defect, datasets and source code quality metrics. Section III explains the proposed platform. Section IV outlines related work, and the last section concludes the paper.

II. FUNDAMENTALS

A. Bug Datasets

Lately, bug datasets are composed for bug or defect prediction. Following this, Ferenc et al. [4] compiled and standardized existing public bug datasets. The same group [5] extended their bug dataset and made the dataset publicly available at [6]. Several research works have produced and utilized bug datasets to develop and evaluate novel bug prediction methods. The objective of their study is to collect and combine current public source code metrics-based bug databases. In addition, they evaluated the abundance of gathered metrics and the bug prediction skills of the unified bug dataset. One research direction in this field moves toward combining bug datasets with software code quality metrics for better prediction. One of the first attempts is published by Osman et al. [7]. They evaluated sixty distinct bug prediction setting combinations on five open-source Java projects using a cost-aware evaluation scheme. Change measurements combined with source code metrics were discovered to be the most cost-effective option for developing a bug predictor. Another example of this work is presented by Mashhadi et al. [8]. They conducted a quantitative and qualitative study on two prominent datasets (Defects4J and Bugs.jar) utilizing 10 common source code metrics, as well as two popular static analysis tools (SpotBugs and Infer), for the purpose of evaluating their capacity to anticipate flaws and their severity.

B. Source Code Quality Metrics

Software quality metrics have been proposed for decades. The literature starts in 1970s. In the 1980s and 1990s, design metrics and their impact on software and source code were mainly studied. Henry and Selig [9] published a book on design metrics, which predicts source code quality. Two early research works specifically on source code quality metrics are by Pearse and Oman [10] and by Welker et al. [11]. They worked on the maintainability of source code.

With the popularity of object-orientation, the research in this area was intensified. Nuñez-Varela et al. [12] did a comprehensive mapping investigation on 226 articles that were published between 2010 and 2015 and discovered nearly 300 source code metrics. Even though object-oriented metrics have received a great deal of attention, there is a need for greater research on aspect and feature-oriented measurements. Prediction of software faults, complexity, and quality evaluation were recurring themes in these investigations.

Currently, there are separate tools as well as tools embedded into platforms, which not only produce source code quality metrics but also calculate technical debt. The next step for these tools seems to be towards predictions and suggestions for better code quality. Our vision and current attempt are in the same direction.

III. PROPOSED PLATFORM

We propose a federated code quality query and analysis platform, called FSCQQA. In this section, we first explain our design goals, such as "authentication and authorization" and "logging and monitoring" and continue with the services the FSCQQA platform provides. Some local services may vary between sites, but standardized procedures and rules will be implemented to ensure uniform administration and oversight. Finally, we present our user interface design to give a sense of use cases for the FSCQQA platform.

A. Design Goals

The major design goals are as follows:

Authentication & Authorization (AA): Each partner or site may have its own AA mechanism implemented. Then, each partner is responsible for the FSCQQA platform for its users'

queries. Each query includes the user and site identification; the site is responsible for logging the queries.

Access Control (AC) Policies: Each site may have its policies and regulations depending on the country where the site is. Therefore, the response to each query is filtered locally before sending. Each site should guarantee that any response does not contain any personal identifiable information.

Secure Communication: Each site must be able to communicate securely with trustworthy peers. All nodes exchange secure Public Key Infrastructure certificates in order to establish trust. While the FSCQQA platform is a federated network, the security of the nodes is only as strong as the network's weakest link.

Logging and Monitoring: Every query executed by a node should be recorded in an audit trail that the peer sites could view. The logs will be monitored by the central site for anomalies.

Standard APIs: Each site should provide standard APIs defined by the FSCQQA platform. Although the FSCQQA platform provides a software agent called FSCQQA agent to fulfil this requirement, the site may choose to implement its own software agent.

Source Code Repositories: The FSCQQA platform provides a software agent to work with GitHub [13] repositories. However, this is not a must. Any site can work with any source code repository but must ensure that standard APIs required by the platform are provided.

Management of Federated Platform: There is a central site responsible for the management of partners and their sites. These management operations include adding and removing partners and sites (a partner may have more than one site), constantly informing partners about other alive partners and sites, and collecting velocity and trend information from site.

B. Services

The FSCQQA platform defines two types of services, one provided by the FSCQQA agent and the other by the standard FSCQQA APIs. The FSCQQA agent is customizable through configurations with the following parameters:

- GitHub repository address
- GitHub repository access rights

The FSCQQA agent automatically generates local defect database for each site from a GitHub repository by extracting commit/issue histories and analyzing them. At the same time, it collects software metrics, such as lines of code and cyclomatic complexity, for each commit/issue. The defect information with software metrics will represent source code quality of the software developed at a site. Moreover, the FSCQQA agent extracts source code related metrics for a specific version using tools, such as OpenStaticAnalyzer [14]. The process is presented as a Unified Modeling Language (UML) sequence diagram in Figure 2. The FSCQQA agent is also responsible for the management of the local database for defects and metrics. To mitigate security concerns related to such an agent software, its source code should be open.

The standard FSCQQA APIs provide the services of the FSCQQA platform with respect to Open-API specifications [15]. The services are grouped as follows:

- Defect related metrics: number of existing (active) defects, defect density, defect resolve velocity, longest unresolved defect.
- Source code related metrics: class metrics, method metrics, coupling metrics, cohesion metrics, cyclomatic complexity metrics.

The services provide data for a specific version. They can be extended to supply data between two versions, but it may complicate the presentation of information and is, therefore, left as future work. The service calls can be for a specific metric or a set of metrics from a specific site or the whole network. If the whole network is queried, the query site requests all alive sites from the central site and queries each one individually then accumulates the results.

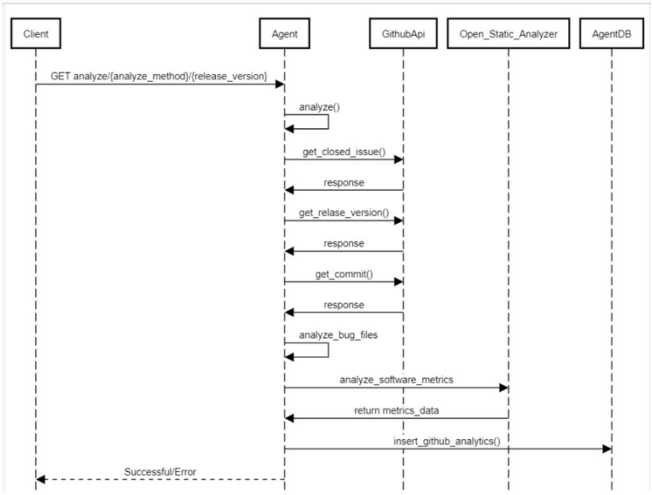

Figure 2. The FSCQQA platform operation.

The central site keeps a list of alive sites in the federated network by recording their heartbeats. Each site is expected to send a heartbeat every hour. If a site's heartbeat is missing necessary notifications are performed. The central site also holds summarized metrics for the whole network, such as overall defect resolve velocity and its trend over some time.

C. User Interface Design

The user interface design is presented via Figures 3-5. A user either in a site or in the central site can see the repositories with proper access rights, as shown in Figure 3. To mimic this operation, Figure 1 presents some public GitHub repositories. This project repository and selection window also indicates the status of the project with four states: "Not Analyzed", "Analyzing", "Analyzed", and "Failed". After selecting a project, a window like the one in Figure 4 is shown and if the status is neither "Analyzing" nor "Analyzed", the "Analyze" button appears. If it has already been analyzed, the details of the analyze operation are shown. To see the metrics, the metrics button should be pressed, and it takes the user to a window like the one shown in Figure 5. It is called the dashboard and presents various

metrics with charts and graphs. Metrics, charts, and graphs are all customizable.

IV. Prototype Implementation

In this section, we present our prototype implementation. The implementation is composed of the following components:

- Web Client: is the main interface for users to interact with the system.
- Master Application: manages a database with details of all agents such as IP and port. It efficiently routes requests from the web client to the appropriate agent and forwards the results back.
- Agent: specializes in executing detailed code analysis tasks, securing sensitive data, and then reporting findings back to the master. Each site is managed by an agent. In this protype implementation, we mimic each site with a GitHub repository.
- GitHub API: is an interface that allows us to access project source codes and general information, facilitating integration with various Github repositories.

The master application has a layered architecture with the following layers:

- Controller: processes various HTTP requests and routes the flow according to the request type.
- Service: performs business operations, coordination tasks, and interaction with the Controller and Repository layers.
- Repository: provides uninterrupted access to the database by facilitating basic data operations.

The master application provides the API endpoints shown in the Appendix. Each endpoint has a controller, and the service layer provides necessary operations with the help of Repository layer.

Each agent registers with the master application before starting any operations. When the user prompts the agent requests repository and project information from GitHub API. Then the agent stores the fetched repository and project information to the database through the master application.

The GitHub repository information contains the following elements:

- Watchers: Indicates the number of users monitoring the repository for changes.
- Topics: Tags or subjects associated with the repository.
- License: Details about the repository's licensing, including its type, URL, and some specific attributes.
- Visibility: Shows if the repository is public or private. In this case, it's public.
- Forks Count: The number of times this repository has been forked by other users.
- Stargazers Count: The number of users who have "starred" the repository, indicating their appreciation or interest.
- Default Branch: The primary branch of the repository, commonly where main development takes place.
- Homepage: The official homepage or documentation link for the repository.
- Number Of Contributor: The number of users who have contributed to the repository.

Figure 3 shows an example of a repository and project versions in that repository. The user can choose a version to be analyzed. As an example, it is 8.0.1.Final version of hibernate-validator in Figure 3. The prototype implementation uses OpenStaticAnalyzer v5.1.0. It provides source code analysis with 46 metrics, such as "Lines of Code", "Comment Lines of Code", "Lines of Duplicated Code", and "Total Number of Statements".

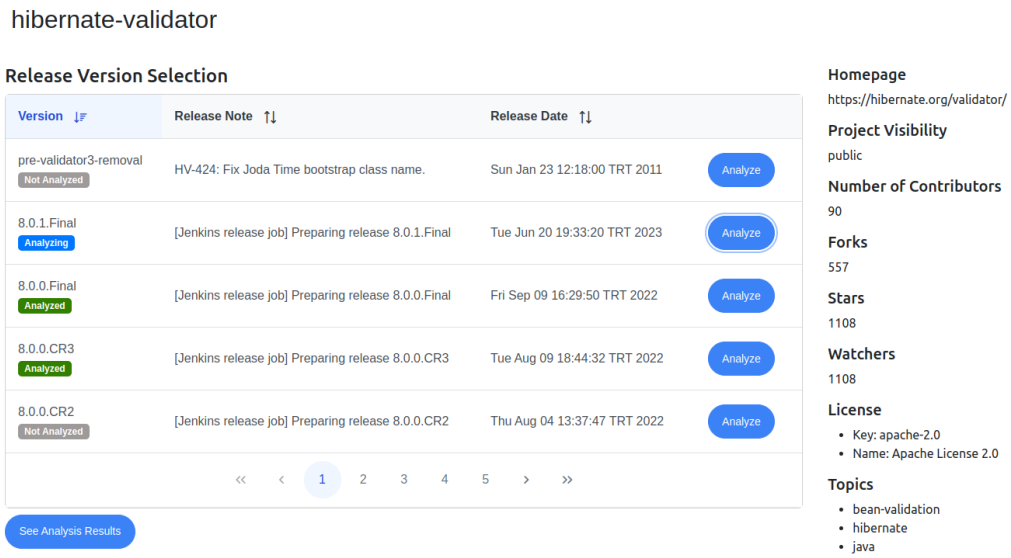

Figure 3. Project repository and release version selection user interface.

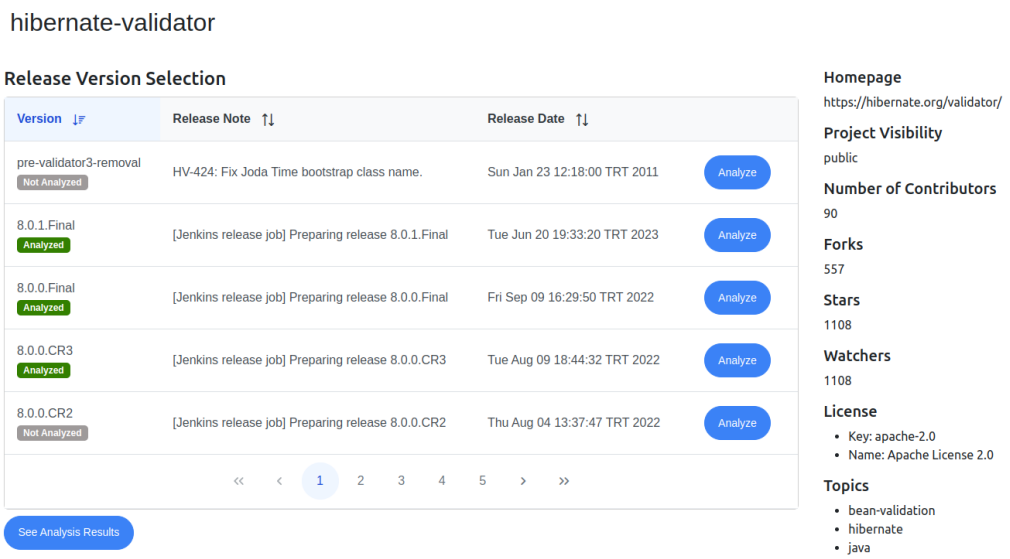

Figure 4. List of analyzed release versions user interface.

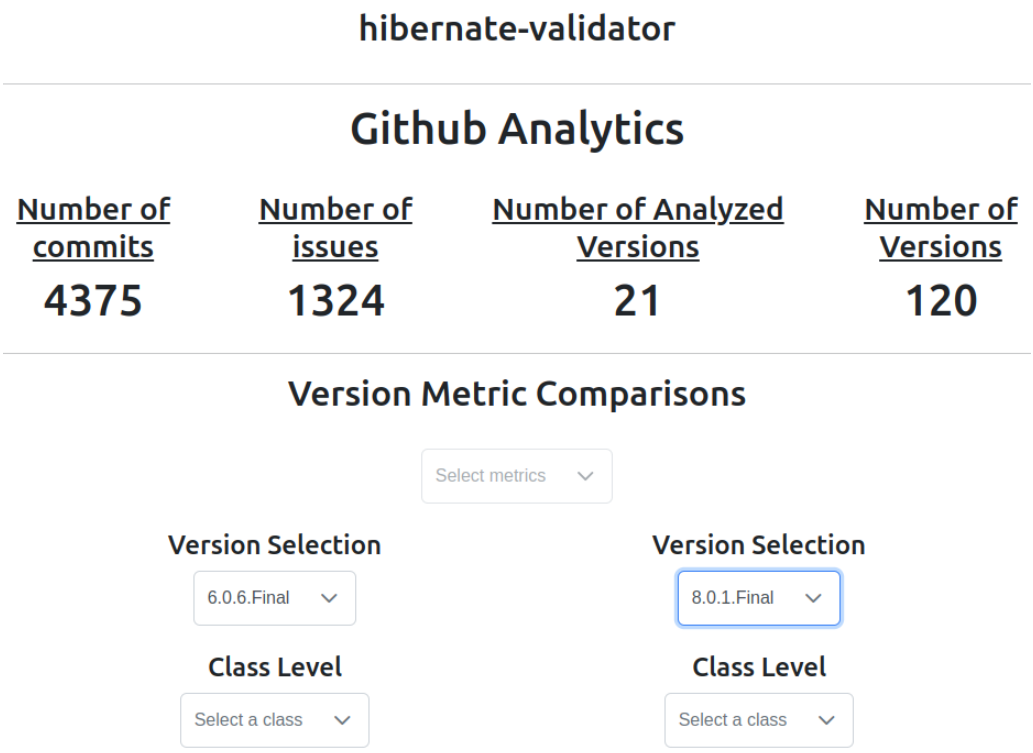

Figure 5. Version metric comparisons user interface.

Figure 4 shows an example list of analyzed release versions. At any time, the user can click the "See Analysis Results" button and the web clients shows a GUI such as in Figure 5. At the top, the following information gathered from GitHub is presented along with the number of analyzed versions:

- Number Of Commits
- Number Of Versions
- Number Of Issues

Analyzed versions becomes important if the user wants to compare versions. The metrics mentioned above are listed as shown in Figure 6. The user must choose minimum three metrics. Selection of metrics up to five is allowed. In addition to metrics selection, the user is asked to choose the classes from two different versions to be compared. As shown in Figure 6, the ReflectionHelper class is chosen as an example.

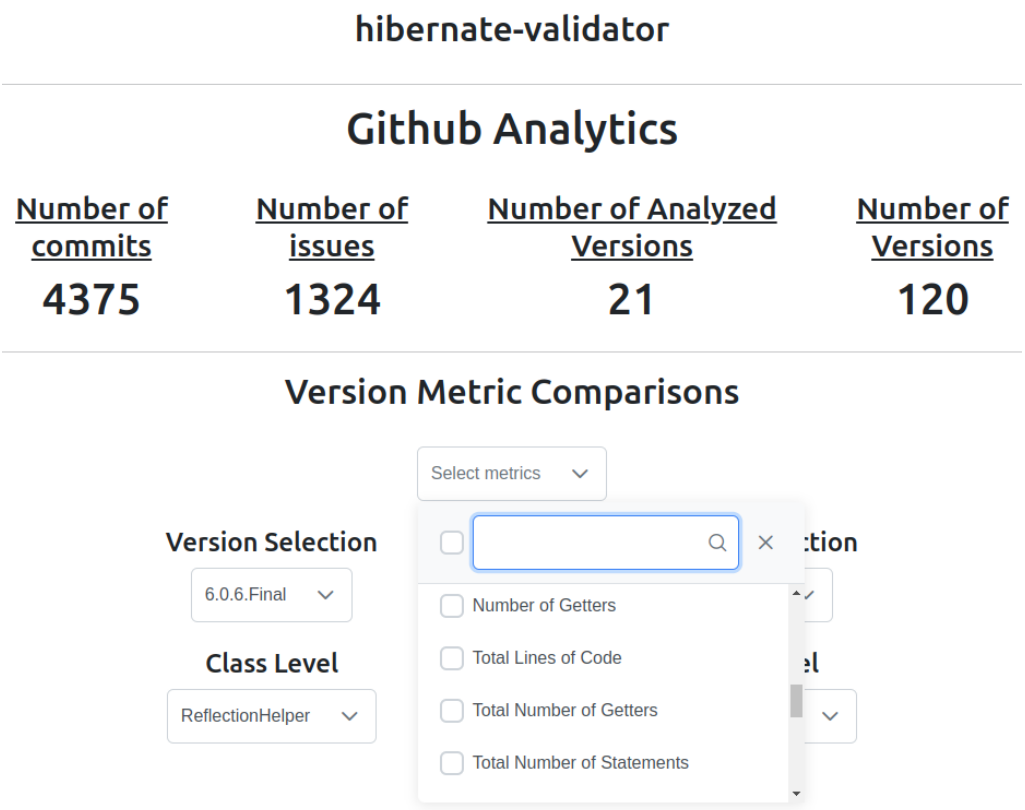

Figure 6. Version metric comparisons user interface with metric and class selection.

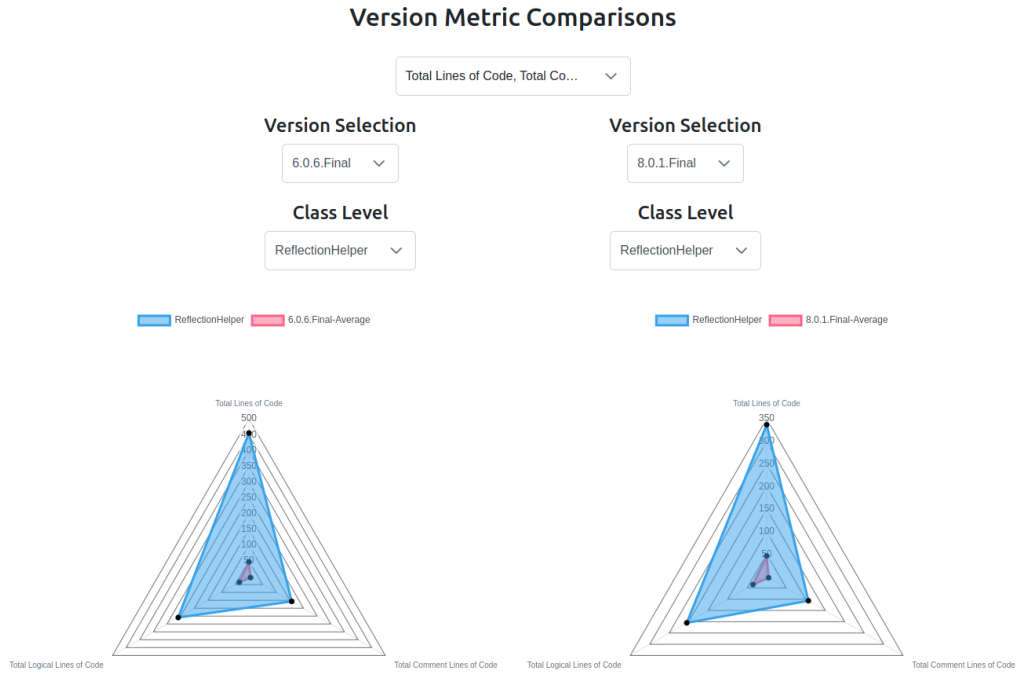

Figure 7. Version metric comparisons user interface with a three-dimensional spider chart.

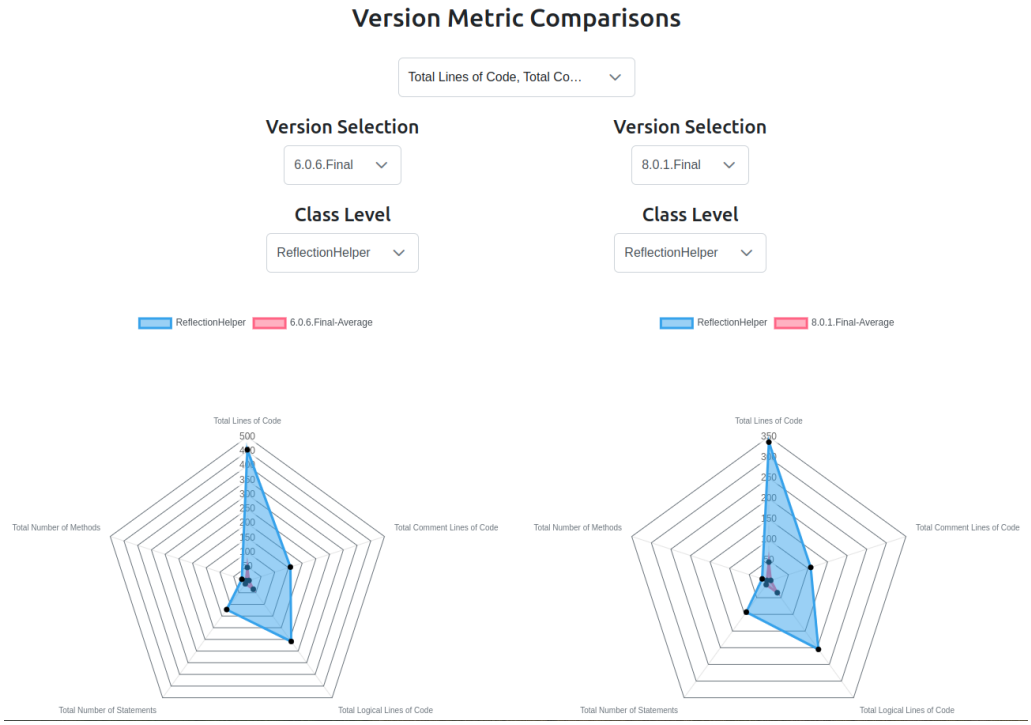

Figure 8. Version metric comparisons user interface with a five-dimensional spider chart.

Figure 7 shows version metric comparisons of the user interface with a three-dimensional spider chart. The dimensions are the chosen metrics. For this example, they are "Total Lines of Code," "Total Comment Lines of Code", and "Total Logical Lines of Code". The spider chart illustrates class metric values and average values for all classes in that version. This presentation technique enables users to compare values intra-version and inter-version. The spider chart is specifically chosen because of the patterns/shapes that appear. They are easier to compare.

The specific example in Figure 7 is indicative in terms of code quality. It is observed that the ReflectionHelper class is larger compared to other classes but throughout the versions it becomes smaller. The shape not being changed means the ratios between the metrics are preserved.

New metrics can be added to the spider chart without effort, as seen in Figure 8. For example, "Total Number of Statements" and "Total Number of Methods" are also chosen, which makes a five-dimensional spider chart. This five-dimensional spider chart makes the change in the ReflectionHelper class more apparent.

The prototype implementation does not contain all the features explained above. It is performed to show the feasibility of the proposed approach and design. This way we aimed to validate the main features of the Federated Source Code Quality Query and Analysis Platform.

V. Related Work

The concept of federated networks is not new, and they are not limited to a certain field. The services are called federated if their service architecture spans numerous independent control domains [16]. It is challenging to manage federated services and provide effective customer assistance since only a tiny portion of the environment can be monitored and controlled by any given authority. Bhoj et al. [16] characterized many facets of federated networks as early as 1997.

Some other examples of federated networks are as follows. For instance, Afsarmanesh et al. [17] proposed the PRODNET architecture for federated information management. Another example is Open Cirrus [18], which is proposed to federate a multitude of sites with diverse hardware, services, and tools for providing federated data centers for open source systems and services research. The sites reside on different continents and are subject to different privacy legislation and concerns.

The health domain is currently running federated networks. For instance, CanDIG [19] is a Canadian national health federated research data platform designed to assist the finding, querying, and analysis of permitted health research data across institutions and projects. CanDIG is the first Canadian federation of many human genomes and biomedical data projects. Another proposal for health domain is the Cross-Institutional Clinical Translational Research project [20], which investigated a federated query tool and examined how this tool can facilitate the discovery of clinical trial cohorts by controlling access to aggregate patient data housed in academic medical centers that are not linked.

VI. Conclusion

Each day, new features are added to software, and with each new feature, extra bugs may be introduced, and source quality may suffer. The scenario becomes more complicated if the software development is distributed with specific privacy and trade secret considerations. When addressing the challenges mentioned above, it is desired that the software quality be maintained above a particular threshold. Toward this goal, this paper proposes a federated source code quality query and analysis platform called FSCQQA.

With the proposed platform, sites are not required to disclose their codes with any other site while aiming for high source code quality and low defect ratio. At each site, local defect datasets will be generated and analyzed. The analysis results as defect metrics and the source code metrics obtained from the static analysis will be shared within the federated network and can be queried. Furthermore, trend analysis can be conducted at the central site and shared with consortium sites.

As future work, federated analytics and prediction using the local datasets are planned. The sites may push defect-related features to the central site for future machine learning. Such a defect database is valuable in terms of following the reliability of each site but also in improving defect-free development by providing in-depth analysis, such as root-cause analysis, and suggesting training and education. Then, the prediction model will generate predictions on sites. The prediction model will be updated and enhanced based on further coming data, meaning new source code. As developer data will not be exchanged, there will be no privacy concerns.

Acknowledgment

The authors would like to thank Univera Inc. for valuable guidance.

References

[1] T. Tuglular, O. Leblebici, E. B. Karaca, N. Uygun, and O. A. Hiçyılmaz, "A Federated Source Code Quality Query and Analysis Platform," presented at the SOFTENG 2023, The Ninth International Conference on Advances and Trends in Software Engineering, Venice, Italy, Apr. 2023, pp. 41–46.

[2] "Networked European Software and Services Initiative-support office team." https://cordis.europa.eu/project/id/034359 (accessed Mar. 19, 2023).

[3] "Methodology and supporting toolset advancing embedded systems quality." https://cordis.europa.eu/project/id/286583 (accessed Mar. 19, 2023).

[4] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for java," presented at the Proceedings of the 14th international conference on predictive models and data analytics in software engineering, 2018, pp. 12–21.

[5] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for java and its assessment regarding metrics and bug prediction," *Software Quality Journal*, vol. 28, pp. 1447–1506, 2020.

[6] "Unified Bug Dataset." http://www.inf.u-szeged.hu/~ferenc/papers/UnifiedBugDataSet/ (accessed Mar. 19, 2023).

[7] H. Osman, M. Ghafari, O. Nierstrasz, and M. Lungu, "An extensive analysis of efficient bug prediction configurations," presented at the Proceedings of the 13th international conference on predictive models and data analytics in software engineering, 2017, pp. 107–116.

[8] E. Mashhadi, S. Chowdhury, S. Modaberi, H. Hemmati, and G. Uddin, "An Empirical Study on Bug Severity Estimation Using Source Code Metrics and Static Analysis," *arXiv preprint arXiv:2206.12927*, 2022.

[9] S. M. Henry and C. L. Selig, *Design Metrics which Predict Source Code Quality*. Department of Computer Science, Virginia Polytechnic Institute and State University, 1987.

[10] T. Pearse and P. Oman, "Maintainability measurements on industrial source code maintenance activities," presented at the Proceedings of International Conference on Software Maintenance, IEEE, 1995, pp. 295–303.

[11] K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and application of an automated source code maintainability index," *Journal of Software Maintenance: Research and Practice*, vol. 9, no. 3, pp. 127–159, 1997.

[12] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, "Source code metrics: A systematic mapping study," *Journal of Systems and Software*, vol. 128, pp. 164–197, 2017.

[13] "GitHub." https://github.com/ (accessed Mar. 19, 2023).

[14] Department of Software Engineering, University of Szeged, Hungary, "OpenStaticAnalyzer." https://openstaticanalyzer.github.io/ (accessed Mar. 19, 2023).

[15] "OPENAPI Initiative." https://www.openapis.org/ (accessed Mar. 19, 2023).

[16] P. Bhoj, D. Caswell, S. Chutani, G. Gopal, and M. Kosarchyn, "Management of new federated services," presented at the Integrated Network Management V: Integrated management in a virtual world Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management San Diego, California, USA, May 12–16, 1997, Springer, 1997, pp. 327–340.

[17] H. Afsarmanesh, C. Garita, Y. Ugur, A. Frenkel, and L. O. Hertzberger, "Design of the federated information management architecture for PRODNET," presented at the Infrastructures for Virtual Enterprises: Networking Industrial Enterprises IFIP TC5 WG5. 3/PRODNET Working Conference on Infrastructures for Virtual Enterprises (PRO-VE'99) October 27–28, 1999, Porto, Portugal 1, Springer, 1999, pp. 127–146.

[18] R. H. Campbell *et al.*, "Open Cirrus™ Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research.," *HotCloud*, vol. 9, pp. 1–1, 2009.

[19] L. J. Dursi *et al.*, "CanDIG: Federated network across Canada for multi-omic and health data discovery and analysis," *Cell Genomics*, vol. 1, no. 2, p. 100033, 2021.

[20] N. Anderson *et al.*, "Implementation of a deidentified federated data network for population-based cohort discovery," *Journal of the American Medical Informatics Association*, vol. 19, no. e1, pp. e60–e67, 2012.

APPENDIX