Software Quality Assessment Tool

Source code evaluation tool for undergraduate and postgraduate projects

Owen Noel Newton Fernando,
Vajisha U. Wanniarachchi,
Yohan Fernandopulle
School of Computer Science and Engineering
Nanyang Technological University,
Singapore

Chaman Wijesiriwardana
Faculty of Information Technology
University of Moratuwa,
Sri Lanka

Prasad Wimalaratne
University of Colombo School of Computing
University of Colombo
Sri Lanka

***Abstract*— Software engineering research is intended to help improve the practice of software development and the quality of the software product. Software quality is one of the critical components in the entire software development process. It is even crucial in assessing the quality of the undergraduate projects. Available software metrics and the tools are mainly focusing on the enterprise level software. Therefore, such tools do not provide means to assess the undergraduate projects critically. This project will concentrate on developing algorithms and introduce a set of metrics that are accurate to evaluate the quality of undergraduate projects. The algorithms and the applications are not only limited to assess the projects. It is modifiable and expandable to evaluate students' assignments submitted by using high-level programming languages such as Java and C++.**

Keywords-software engineering; software quality attributes; software quality metrics

I. INTRODUCTION

Manual inspection and review of the source code of undergraduate/postgraduate projects are experiencing several critical problems such as:

- It requires a considerable investment in preparation for the review and evaluation of the projects.
- It requires the contribution of the supervisor as well as a programming language expert.
- Human code auditors must be initially aware of what types of errors are supposed to find before they can rigorously examine the code.

However, such a critical assessment of the software projects is essential to prepare students to work on realistic large-scale software projects in the industry. It is often difficult to ensure a right balance between academic and software industrial concerns and to create assessments that adequately address the skills and knowledge requirements of both sectors.

Static analysis tools compare favorably to manual reviews because they perform faster, and they encapsulate some of the knowledge required to carry out this type of code analysis by a human auditor. Static analysis can decrease the amount of testing and debug necessary for the software to be deemed ready. However, existing static code analysis tools focus on assessing the quality of the source code of enterprise level software. Therefore, such tools cannot be directly adopted to measure the quality of the undergraduate/postgraduate projects due to several reasons.

In the software industry, refactoring and code inspections are necessary quality assurance activities for enhancing the quality of code. In academia, such activities are rarely taught and practiced at Undergraduate level due to various reasons. These reasons may include time constraints, limited knowledge of the available tools and flexibility with the course syllabus.

Software quality, as defined by IEEE [1], is the degree to which a software possesses the desired combination of software quality attributes. These quality attributes have been identified in many standards and models, such as McCall's Quality Model, Boehm's Quality Model, etc. [2]. Since the quality attributes are qualitative, quality metrics are needed to quantify the quality attributes [3]. Therefore, this paper proposes a Software Quality Assessment Tool (SQAT) that automatically assesses the quality of the source codes of undergraduate and postgraduate projects with the use of structural metrics, i.e. object-oriented metrics and coding standards. After evaluating the source codes, the application will produce reports that consists of quality scores, warning messages, locations of bad codes, and suggestions.

II. RELATED WORK

Human analysis of the source code is known as program understanding, program comprehension, code reviews, software inspections or software walkthroughs. However, the human-based code analysis is an age-old concept from the time of discovery of computers.

The term ‘Static code analysis' is usually applied to the analyses performed by an automated tool. Software metrics and reverse engineering can describe as forms of static analysis. Deriving software metrics and static analysis are increasingly deployed together, especially in the creation of embedded systems, by defining software quality objectives [4].

Software quality metric measures some properties of a software system [5]. By measuring software quality metrics, we can evaluate software development process, receive earlier feedback during the development, and assess the progress of software development [6].

Many software quality metrics have defined in various researches. Saraiva et al. [7] have classified 570 software quality metrics related to maintainability. However, previous works have shown that many software quality metrics are lacking a theoretical basis [8] or being too labor-intensive to collect [9]. To avoid these problems Chidamber and Kemerer [10] has listed six design metrics with a theoretical base, i.e. weighted methods per class, depth of inheritance tree, the number of children, coupling between object classes, the response for a class, and lack of cohesion in methods. Harrison et al. [11] also listed six valid Metrics for Object-Oriented Design (MOOD), i.e. method hiding factor, attribute hiding factor, method inheritance factor, attribute inheritance fact, coupling fact, and polymorphism factor.

A software quality metric can relate with multiple software quality attributes, for example, "number of attributes" can be related to reliability, extensibility, reusability, readability, flexibility, traceability, and scalability [7]. The difficulty in measurement of software quality is the relationship between software quality metric and software quality attributes cannot be defined and must be partially imperfect [12]. Also, different types software has different quality requirements, for example, reliability is important for a mission-critical project, but is not essential for consumer-based mobile application. As a result, various software projects must define different quality objectives.

A systematic way to relate software quality metrics with software quality attributes is necessary to measure software quality. Basili and Weiss [13] has developed a goal oriented data collection for collecting valid software engineering data. His work was later used by YI [6] to describe Goal Question Metric (GQM) Paradigm, a mechanism for defining and interpreting operational and measurable software. Previous works by Cavano and McCall [12] and Baggen et al. [14] developed ideas that are similar to GQM to relate software quality metrics with software quality attributes, but these ideas are not well known and defined by the authors. Therefore, this research uses GQM to link software quality metrics with software quality attributes.

This project considered framework and structures introduced by Cavano and McCall [12], Baggen et al. [14], Tjoa et al. [15] and Washizaki et al. [5] to choose the most suitable framework for SQAT. Among the considered frameworks, Cavano and McCall presented a framework for large-scale and critical systems. While Baggen et al. considered only the maintainability quality attribute, the Tjoa et al. considered only security quality attribute. Washizaki et al. have proposed a framework that achieves effective measurement and evaluation of source code quality. The framework consists of a comprehensive quality metrics suite, a technique for normalization of measured values, an aggregation tool, a visualization tool for the evaluation of results, a tool for deriving rating levels and a set of derived standard rating levels.

The project further analyzed the performance of two main Software Quality Measurement Tools namely SonarQube [16] and SciTool [17]. SonarQube is an open source continuous code inspection tool. It possesses functionalities such as code checking, code duplication detection, code complexity measurements and some simple metrics which are also related to the objectives of SQAT. But, this tool does not provide automation which is a necessity to achieve the identified objectives of SQAT. It also does not quantify the source code using comprehensive software quality metrics suite. SciTool is an integrated development environment (IDE) with many static code analysis tools. The IDE do have a comprehensive coverage of different programming languages, such as ADA, C/C++, C#, FORTRAN, Java, and VHDL. But, This IDE is an expensive commercial product and mostly used for commercial purposes. Therefore, building a software quality measurement tool can fill these gaps specially to facilitate educational institutes.

III. Design and Implementation

The proposed application expects to handle a large number of requests per day and therefore need a scalable solution to handle the requests. As a solution, this project used Microservices architecture to scale the proposed application horizontally to handle more requests at a time. The proposed

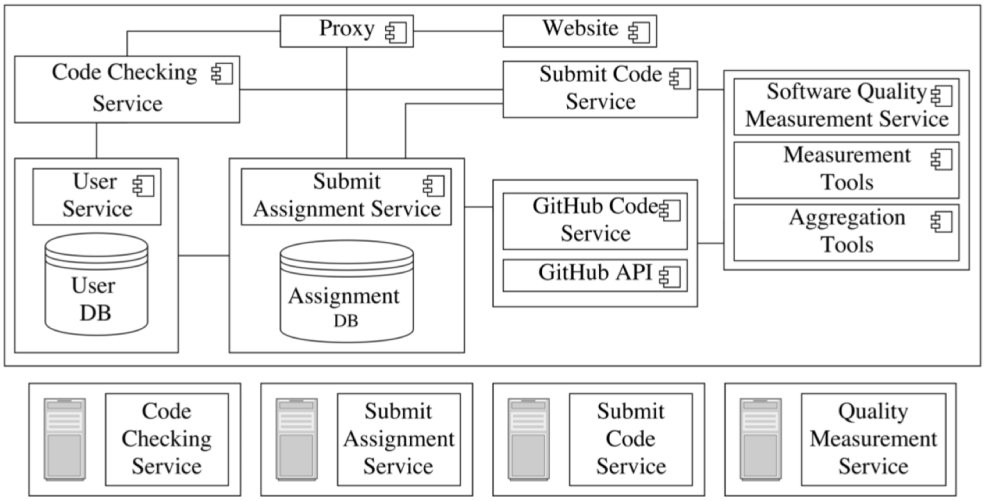

Figure 1. Microservices Architecture for SQAT

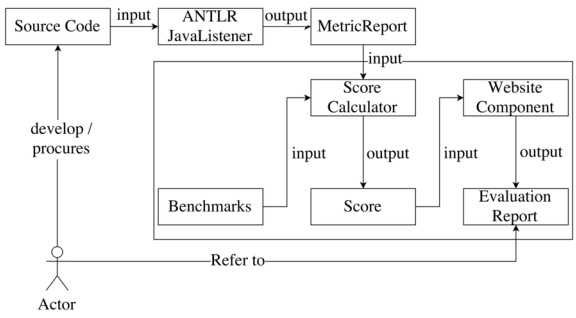

Figure 2. Structure of the SQAT quality measurement service

application has developed as four services namely; code checking service, submit assignment service, submit code service and quality measurement service (see Figure 1), to cater the objectives of this project. Since the Remote Procedure Call (RPC) library developed by Google (GRPC) used Protocol Buffer to serialize structure data in a language agnostic way, SQAT uses GPRC to connect the defined services. While the code checking, submit assignment and submit code services handled the assignment and code submission process, quality measurement service act as the core component of the SQAT.

A. Architecture of Quality Measurement Service

As the core component of the SQAT, this service measures software quality attribute quantitatively using Goal Question Metric (GQM) paradigm. The framework proposed by Washizaki et al. [5] is used to implement this service (see Figure 2).

1) Code Style Configuration

Consistent code styles are important for maintainability of software projects. The software measurement component enforces a small subset of code styles. For example, the indentation levels, import statement styles, and method name format must be consistent with a piece of codes. However, not all projects have the same set of code style requirements; some projects might want to use two space characters for indentation, and some projects ought to use four space characters. As a result, code style for each project must be configurable.

The software measurement component does accept code style configuration as an argument in its API. Currently, the component only allows configuration written in Javascript Object Notation (JSON) format.

2) ANTLR

After the user submits the source code of their software/project, it inputs to the ANTLR JavaListener. The ANTLR is a powerful parser generator for reading, processing, executing, or translating structured text or binary files (Parr, 2013). The SQAT uses ANTLR to collect metrics from Java codes in software measurement service. Since a Java language structure should define in a grammar file which will be accepted by ANTLR, an open source Java language definition Grammar-V4 project [18] is used. Gradle [19] and ANTLR plugin for Gradle are used to generate JavaLexer, JavaParser, and JavaListener. To analyze and collect metrics from a Java code chunk, JavaLexer is used to produce a stream of tokens and JavaParser is used to generate a parse tree. The ParseTreeWalker is used to traverse the generated parse tree and while traversing the Walker will inform JavaListener which has defined by the authors of this paper. The output of the ANTLR will feed to generate the MetricReport.

TABLE I. USING GQM TO BUILD QUALITY METRIC SUITE

Goal	Question	Metrics
Analysability	Is the size of the code not too large?	Line of codes
	Are the conditional statements not deep?	Depth of conditional nesting
	Are the naming of variables good?	Average length of identifier
	Are classes too complicated?	Number of Attributes
	Are classes too complicated?	Number of Methods
Testability	Is the size of the code not too large?	Line of codes
	Are the conditional statements not deep?	Depth of conditional nesting

3) The Score Calculator

The Score calculator normalizes measured values to calculate the score. This project uses the rating level deriving tools to derive benchmark values, to compare the measured values with some benchmarks values. The benchmark values and the measured values fed into score calculator, which gives the final score of the source code from the component level up to the whole system. However, it only allows evaluation of score at the component level, which is different from the aggregation tool defined by Washizaki et al. [5] that allows evaluation from the component level up to the whole system.

The GQM paradigm is used to build the quality metric suite. In the initial version of SQAT, two goals and 5 number of question and metric pairs have defined. The quality metric suite is shown in Table I. This quality metric suite will be used by score calculator to calculate the score for a given source codes.

Let say the benchmark value for depth of conditional nesting is equal to one. If we found out that the depth of conditional nesting of a piece of code is equal to two, what score should we give to this software metric?

Washizaki et al. [5] solve this problem using linear piecewise functions. Specifically, if the collected value is less than or equal to benchmark value, the score for the software metric is 100%. Else if the collected value is more than benchmark value and less than three times of benchmark + upper hinge, the score will decrease according to the following equation 1:

$$score = \left(-\frac{1}{3\times benchmark}\times value + \frac{4}{3}\right)\times 100\% \quad (1)$$

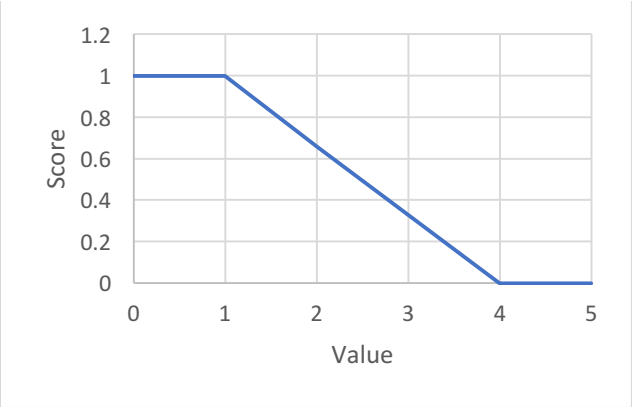

Figure 3. Example of score calculation graph for a software metric with benchmark value equal to one

Different metrics have different severity level depending on their impact to application reliability, thus the score need to be penalised accordingly. The equation 2 is used in this project.

$$score = \left(-\frac{1}{severity * benchmark} * value + \frac{severity+1}{severity}\right) * 100\% \quad (2)$$

Any value that is greater than three times of benchmark + upper hinge will give a score of zero. The upper hinge is equal to benchmark in our case. The linear piecewise function has plotted in Figure 3. To answer the question which we present at the beginning of this section, we just need to substitute benchmark = 1 and value = 2, and we should get the score = 66.67%. The linear piecewise function to calculate scores for a software metrics can implement easily.

After calculating the score for each metric, the calculation of scores for software quality attributes become trivial. We just need to take the average of software metrics that are related to a software quality attribute, as defined in the software quality metric suite in Table I. We show an example in Table II. In this example, the analyzability score would be (80 + 90 + 70)/3 = 80% and the testability score would be (80 + 90)/2 = 85%.

Finally, the visualization tool has mapped to a web component in SQAT, which will show the result in a single page application. The component is developed using Flux Architecture. The React.js which promotes the compostable and reusable user interface components is used to develop the user interface. The Alt.js library is used to manage of the web component.

TABLE II. CALCULATED SCORED OF SOFTWARE QUALITY ATTRIBUTES USING GQM

Goal	Question	Metrics	Score
Analysability	Is the size of the code not too large?	Line of codes	80%
	Are the conditional statements not deep?	Depth of conditional nesting	90%
	Are the naming of variables good?	Average length of identifier	70%
Testability	Is the size of the code not too large?	Line of codes	80%
	Are the conditional statements not deep?	Depth of conditional nesting	90%

IV. DISCUSSION AND FUTURE WORK

This project can be further improved in five different ways, i.e. conducting acceptance testing, implementing a rating level deriving tool, developing a better software quality metric calculator, developing a better score calculator, and supporting more configuration file format.

Although we only have an initial prototype of the tool, conducting an acceptance testing earlier would allow us to detect flaws in our requirements and functionality earlier. By doing so, we can eliminate bugs and set the development road map of SQAT in the right direction. The rating level deriving tool was initially mentioned in Washizaki et al. [5].

It should be a tool to extract software quality metrics from a given software project in a particular programming language. By having this tool, we can collect software quality metrics data in scale, and hence, able to develop benchmark values for software quality metrics.

The aggregation tool is described in Washizaki et al. [5] paper as a tool that allows evaluation of software quality from the component level up to the whole system. However, the score calculator in SQAT only allows evaluation of software quality at the component level only. By improving this tool, we can zoom in to evaluate at the component level and zoom out to evaluate at packages and whole system level.

The Goal Question Metric (GQM) approach is essential for SQAT. The GQM in SQAT should be improved to cover more software quality attributes and more metrics that related to these attributes. By doing so, we can get better analysis reports.

V. CONCLUSION

The project has developed the foundation of software quality measurement tool named as SQAT. The main goal of this project is to build an automatic code assessment tool. The tool will be used to analyze codes for projects and assignments of undergraduate students. Since the defined software quality attributes are qualitative, there is a necessity to discover a method to qualify the qualitative attributes. Therefore, this project used GQM approach to calculating the scores for a software quality attributes. The quality measurement component is implemented based on the framework proposed by Washizaki et al. [5]. A scalable architecture named Microservices is used to develop the proposed application. ANTLR is used as the core tool to analyze the submitted codes and collect software metrics. The website which is implemented as the front-end for project submission is developed using Flux Architecture.

REFERENCES

[1] "IEEE Standard for a Software Quality Metrics Methodology", 1998.

[2] P. Berander, L. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jönsson, S. Kågström, D. Milicic, F. Mårtensson, K. Rönkkö and P. Tomaszewski, *Software quality attributes and trade-offs*, 1st ed. Blekinge Institute of Technology, 2005.

[3] E. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster and P. Avgeriou, "A mapping study on design-time quality attributes and metrics", *Journal of Systems and Software*, vol. 127, pp. 52-77, 2017.

[4] T. Cambois and P. Munier, "Software Quality Objectives for Source Code", MathWorks Automotive Conference, 2010.

[5] H. Washizaki, R. Namiki, T. Fukuoka, Y. Harada and H. Watanabe, "A framework for measuring and evaluating program source code quality", in *International Conference on Product Focused Software Process Improvement*, 2007, pp. 284-299.

[6] V. Caldiera and H. Rombach, "Goal question metric paradigm", *Encyclopedia of Software Engineering*. pp. 528-532, 1994.

[7] J. Saraiva, M. de França, S. Soares, F. Filho and R. de Souza, "Classifying metrics for assessing Object-Oriented Software Maintainability: A family of metrics' catalogs", *Journal of Systems and Software*, vol. 103, pp. 85-101, 2015.

[8] I. Vessey and R. Weber, "Research on Structured Programming: An Empiricist's Evaluation", *IEEE Transactions on Software Engineering*, vol. -10, no. 4, pp. 397-407, 1984.

[9] C. Kemerer, "Reliability of function points measurement: a field experiment", *Communications of the ACM*, vol. 36, no. 2, pp. 85-97, 1993.

[10] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.

[11] R. Harrison, S. Counsell and R. Nithi, "An evaluation of the MOOD set of object-oriented software metrics", *IEEE Transactions on Software Engineering*, vol. 24, no. 6, pp. 491-496, 1998.

[12] J. Cavano and J. McCall, "A framework for the measurement of software quality", *ACM SIGSOFT Software Engineering Notes*, vol. 3, no. 5, pp. 133-139, 1978.

[13] V. Basili and D. Weiss, "A Methodology for Collecting Valid Software Engineering Data", *IEEE Transactions on Software Engineering*, vol. -10, no. 6, pp. 728-738, 1984.

[14] R. Baggen, J. Correia, K. Schill and J. Visser, "Standardized code quality benchmarking for improving software maintainability", *Software Quality Journal*, vol. 20, no. 2, pp. 287-307, 2011.

[15] S. Tjoa, P. Kochberger, C. Malin and A. Schmoll, "An Open Source Code Analyzer and Reviewer (OSCAR) Framework", in *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, 2015, pp. 511-515.

[16] "Continuous Code Quality | SonarQube", *Sonarqube.org*. [Online]. Available: https://www.sonarqube.org.

[17] "SciTools.com", *Scitools.com*. [Online]. Available: https://scitools.com.

[18] "antlr/grammars-v4", *GitHub*. [Online]. Available: https://github.com/antlr/grammars-v4/blob/master/clojure/Clojure.g4.

[19] "Gradle Build Tool", *Gradle.org*. [Online]. Available: https://gradle.org.