ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 171-175
doi: 10.18178/wcse.2017.06.030

SonarQube as a Tool to Identify Software Metrics and Technical Debt in the Source Code through Static Analysis

Daniel Guaman[1 2+], Pablo Alejandro-Quezada Sarmiento, [1 2 3] Luis Barba-Guamán[1], Paola Cabrera[1], Liliana Enciso[1]

Departamento de Ciencias de la Computación y Electrónica, Grupo Innovación Tecnológica Educativa (GITE), Departamento de Ciencias de la Educación, Escuela de Informática y Multimedia
[1] Universidad Tecnica Particular de Loja, [2] Universidad Politécnica de Madrid ,[3] Universidad Internacional del Ecuador; Loja –Quito- Ecuador

Abstract. Technical Debt (TD), also known as technical debt design or technical debt code, analyze the consequence that could have a system once it has been designed architecturally, coding or implemented. TD refers to work to be performed rather than software design or coding is considered complete or correct. Static analysis is a technique to identify and analyze software characteristics from source code; through static analysis we can identify elements such as packages, classes, relationships, lines of code (LOC's), bugs, complexity, coding violations and others. In addition, subsystems, components and their relationships supported by tools, algorithms, frameworks to analyze the code were identified. SQALE is a quality and analysis model contains the internal properties expected from the code in the context of the evaluation, it has been used to perform many assessments of software source code, of various sizes in different application domains and programming language. SonarQube is an open source platform to manage the source code quality, this cover seven axes of code quality among which stand: architecture and design, duplications, unit test, complexity, potential bugs, codifications rules, comments, among others; this platform work with over 20 programming languages.
This paper, use as input the source code of the software applications written in different programming language for through static analysis identify metrics, characteristics, and technical debt with the aim to improve the quality when writing code, also supported in static analysis identify aspects such as correct apply of quality attributes, standards and best practices of programming that based in ISO 9126 and SQALE ensure the correct software development in terms of design and coding.

Keywords: Quality attributes, Source code, SonarQube, SQALE, Static Analysis, Technical Debt.

1. Introduction

Most software applications in the companies, organizations, or academic institutions have been developed and designed following a software process [20] [22] [23]; but these applications not always have characteristics, metrics, attributes and other standards in coding or design required to ensure the quality of software in production [21] or in Agile projects, causing software technical debt. TD is associated with extreme programming (XP), a software methodology, in the refactoring context and it was introduced by Ward Cunningham, a programmer known as the first to develop a wiki. This paper exposes the search, classify and analysis of aspects related to technical debt in the code, also showed the use of SonarQube combined with static analysis as a technique to analyze the software characteristics from the source code to determine possible issues in the people that develop software.

The aim of this paper research is analyze if the people use standards, recommendations and best practices to coding software and if do not use how it could affect the complexity, security, bugs, technical debt in the building software. This paper has been structured as follows: Section I, contains the Introduction, section II

+ Corresponding author. Tel.: + 593991066491.
E-mail address: daguaman@utpl.edu.ec.

introduces a Background, section III describes the study design case, in Section IV some conclusions are showed. Then Section V the acknowledge are present, Section VI Future Work (The following steps in this research) is showed.

2. Background

Technical Debt. - TD was introduced at OOPSLA by Ward Cunningham [15] in 1992, and it is related with failures in the software internal quality produced by bad practices in the development or design process, it is always a risk for the future software; for example, start the coding without clear scope, it produced a debt situation which affects interests (overrun) not only in the maintenance process but also in the operation of the application [1]. TD is not related with code and their intrinsic qualities but with structural or architectural options or digital gaps [2].

TD has associated interest's payments specified in the extra effort needed in the future produce by a quick and fast choice [3]. In case of exists TD, we could choose two options:

1. Leave the software application in this conditions without changes and assume interest payments.
2. Consider reducing the debt to fix a specific problem or introducing in the maintenance process minimum evolution objectives in each new version of the software application.

Some authors mention that quality issues in code are considered as TD, without specify details of the quality issues, but there are factors as duplicate code, complexity, issues that affect a fragment of a function or a method [12].

SQALE. - The SQALE Method has been developed to support in the software evaluation; is a generic method independent of a language programming and tools for source code analysis. Also, this method contains a model for estimating source code TD according to implementation technologies and the tailoring of the project [13]. SQALE is implemented through some static analysis tools [14] that produce indices and indicators such as: Clearly define what creates technical debt, correctly estimate this debt, Analyze the debt from technical, business view; and Offer different prioritization strategies to establish the optimize recovery plan [6].

SonarQube. - Is a platform to manage and control the code quality in seven axes through SQALE Method [5, 16]. SonarQube is basically the fusion of two static code analysis tools such as Check Style [7] and PMD [8]. Check Style is a tool of code quality to validate the building standards and find potential problems in the source code and PMD is a source code analyzer; through this tool we can find common programming defects [17].

SonarQube use the quality model SQALE because technical debt and code quality are two connected concepts; through SonarQube run the code and use remediation functions to work out remediation costs for each code element, TD is identified as the sum of remediation costs for all non-compliances.

Static Analysis. - Techniques to analyze code such as clustering, static and dynamic analysis have a great impact on quality attributes like maintainability, verifiability, flexibility, portability, reusability, interoperability, and expandability; to understand the architecture of a system application [11].

The aim of static analysis is to extract the architectural model, and characteristics from the source code, to recover the system structure, to build the system relationships, to analyze the dependence between components, to identify calls to functions into the source code, and to detect coding problems [9].

Static analysis permit identifies elements like files, classes, hierarchical classes, methods, variable's names, packages diagrams, patterns and their relationships, some architectural styles, and coding styles; significant architectural elements could be graphically represented. In addition, subsystems, components and their relationships and well as interactions; through static analysis were identified as example weights of class relations, static coupling, and static cohesion [10].

3. Study Design Case

The TD in the source code is the first most studied because there are several static tools supporting the identification, measurement and repayment of this type of TD, another reason is that the code is concrete and easy to understand and finally there are developers working everyday on code and they have much experience in this kind of activities. Based in code analysis as the main approach of TD identification, in this research, we first apply a survey that contained some questions focuses in the use of coding standards, best

practices and safety recommendations to build software in different programming languages, with the objective of identify if the students apply standards in their software applications; the survey was applied at 100 students of System Engineering in the UTPL. The following activities were:

1. Search information about tools such as SonarQube, Sonar Graph, and Structure Analysis for java to combine with static analysis and identify source code characteristics and metrics to extract data.
2. The students in pairs or individually written code using programming language such as java, php, visual studio, PL-SQL to build their applications.
   The type of applications transactional and non-transactional built were for instance coding java classes project in android studio to interact with Sphero Robot; create an application to vehicle management written in PHP, Google Api's and Mysql as database and Store procedures to interact with JPA; Rest Service in java to obtain academic data of students in UTPL using MVC and Fa çade; a software Web application (written in PHP-MVC) and client (written in java packages) to coffee production in South Zone Equator, and finally a class diagram (four classes, generalization, association) to solve an specific problem in java interacting with a database to make a basic CRUD (Create, Read, Update, Delete). The source code of 40 applications written by students was composed between 200 and 18000 LOC's
3. Source code of each applications organizing it by folders, files, packages, and running each module to load in SonarQube to extract the metrics to analyze code duplications, bad practices, complexity and TD

The principal metrics to analyze are duplication code, complexity, potential issues and TD.

- Duplication code: Detect repeated code expressed in blocks of code, files, number of lines, and percentage of duplicates lines. The percentage of duplicate lines are expressed as *DP = (Number of duplicates lines/Total number of LOC's) *100*
- Complexity: Analyze the code from the logic point view, using the reserved or keywords words of each programming language.
- Potential issues: Are the number of bad practice in writing code. The issues classification is given in terms of "severities" expressed as:

a. Blocker: Error with a high probability of impact on the behavior of applications in production.
b. Critical: Error with a low probability of impact on the behavior of the application in production or an issue that represents a security breach.
c. Major: Quality defect that can have a high impact on developer productivity.
d. Minor: Quality defect with a slight impact on the productivity of developers; for instance, lines that should not be so long.
e. Info: The code does not own any error or defect quality, only a finding. It is not known or is not yet a risk to safety or impact on productivity.

Based on the source code of applications the 37, 5% out of 40 applications contains duplicates lines and 40% out of 100% applications contains duplicate code in blocks and files because of duplicated lines.

It implies that, the code duplications could be controlled making meetings between the developers or review code in group, also using management configuration as a technique to reduce problem in coding, especially in agile projects (Table I).

TABLE I: Lines of code (LOC's) and their relations with function, classes, statements and accessors

LOC's	Functions	Classes	Statements	Accessors	Duplicate Lines	Duplicate Blocks	Duplicate Files
< 1000	267	60	3046	85	756	39	9
> 1001 < 1500	240	48	2945	160	1164	43	10
> 1500	2059	233	25443	813	13324	726	134
63457	2566	341	31434	1058	15244	808	153

TABLE II: complexity

LOC's	Function	Class	File
< 1000	2,87	16,9	17,4
> 1001 < 1500	2,60	13,2	13,2
> 1500	2,80	24,77	17,84
63457	2,76	18,29	16,15

In the form early phases, the duplication code, the growth is exponential; specially we should take care when work interacting with a database because use statements that only could change certain parameter to

obtain the same information. The result shows that if we are working programming daily monitoring, and following a standard documentation the duplicate code is low. The duplications as usual, are independent of LOC's but always depend of best practices, and skills of programmers. Defect in TD refer to poorly written code that violates best coding practices or coding rules, bugs, failure in software system, it produces complex code that expose in the table II.

According the cyclomatic complexity, while more files, LOC's and statements used in the code, the complexity of software increases and more even if exist duplicate code. The complexity is associated to Functions, Classes and Files used in the coding and considered as part of LOC's. The value expressed as 2,76 means that in Function level have a low risk, in Class the value 18,29 expressed that the software is complex considering high risk program. The value 16,15 in the File column exposed that the applications have a moderate risk. To analyze the complexity, use the values exposed in the table III.

TABLE III: complexity

Range Values	Risk Evaluation
1 – 10	Simple program, without much risk.
11 – 20	More complex, moderate risk.
21 – 50	Complex, high-risk program.
50	Program does not testable, Very high risk

Table IV: Potential Issues

LOC's	Issues	Blocker	Critical	Major	Minor	Info
< 1000	1305	0	85	453	740	27
> 1001 < 1500	1398	0	13	333	1023	29
> 1500	8169	0	612	3601	3638	325

The table 4 show the Issues as the principal problems in the software, it could be fixed through interactions, the principal issues are for incorrect uses of variables, assignments, constants and bad use of control statements into the code; the idea is reducing this kind of problems that affect the maintainability, performance, security, testability and other quality attributes in software applications.

4. Conclusions

Based in the results exposes in the above tables, is advisable use standards, quality models and best programming practices improve in maintainability, security, changeability, reliability and testability as part of technical debt pyramid. In case of obviate this best suggestion, we can obtain a high technical debt ratio considering the type of application in terms of complexity and functionally.

The issues Major and Critical are associated to bad practices in programming, that affect directly to TD; have a SQALE Rating classified in A show that we improve the coding in a time suggest for SonarQube, in this case and applying standards try to fix issues, files, duplication. The principal idea of use SonarQube as static analysis tool is to maintain a reference metrics to suggest the developers the use of standards and not increase the TD specially when working in business applications where the working teams are numerous.

For some applications in the present research used a guided developer and working in four or five iterations, for each one using Sonarqube to analyze the source code, if exist problem, fixed in a time given, after that review the recommendations in the tool and proceed again to fix, we need at least four or five iterations to obtain a SQALE Rating A and Technical Debt less than 0,5%, simulating the code review as an Agile Methodology.

5. Acknowledgment

The authors express their gratitude to the Technical University of Madrid (UPM), Doctoral Program of Science and Technology of Computing for Smart Cites, and Department of Information Systems UPM CITSEM.

6. Future Works

As future works we are working over ATD (Architectural Technical Debt) using Static, Dynamic and Clustering Analysis following ATAM Framework and software quality measurements such as ISO 9126, ISO 9126-3.

7. References

[1] A. Mamun, M. A., Berger, C., & J. Hansson," *Explicating, understanding, and managing technical debt from self-driving miniature car projects. In Managing Technical Debt (MTD)*", IEEE. Sixth International Workshop on (pp. 11-18), September 2014.

[2] E. Alzaghoul, & R Bahsoon.," *Economics-Driven Approach for Managing Technical Debt in Cloud-Based Architectures. In Utility and Cloud Computing (UCC)*", IEEE/ACM 6th International Conference on (pp. 239-242). IEEE, December 2013.

[3] E. Alzaghoul, &, R Bahsoon. "*Clouted: Using real options to manage technical debt in cloud-based service selection*", In Managing Technical Debt (MTD), 2013 4th International Workshop on (pp. 55-62). IEEE, May 2013.

[4] M. Von Detent, M. Meyer, & D. Travkin*," Reverse engineering with the reclipse tool suite*", In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2 (pp. 299-300). ACM, May 2010

[5] Available: http://www.sonarsource.com/products/features/technical-debt-evaluation/

[6] Available: http://www.sqale.org

[7] Available: http://checkstyle.sourceforge.net/index.html

[8] Available: http://pmd.sourceforge.net/pmd-5.0.0

[9] T. Arias, B. C., Avgeriou, P., America, P., Blom, K., & Bachynskyy, S. "*A top-down strategy to reverse architecting execution views for a large and complex software-intensive system: An experience report*". Science of Computer Programming, 76(12), 1098-1112, 2011.

[10] C. Chong, Y., Lee, S. P., & T. C. Ling," *Efficient software clustering technique using an adaptive and preventive dendrogram cutting approach*". Information and Software Technology, 55(11), 1994-2012.

[11] I. Pashov, & M. Riebisch "Using feature modeling for program comprehension and software architecture recovery". In Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the (pp. 406-417). IEEE, May 2014.

[12] R. Marinescu, "*Assessing technical debt by identifying design flaws in software systems*", IBM Journal of Researchand Development 56(5) (2012)9:1–9:13.

[13] Z. Li, P Avgeriou & P. Liang, "A *systematic mapping study on technical debt and its management*". *Journal of Systems and Software*, *101*, 193-220, 2015

[14] J. Letouzey, L., & Ilkiewicz, M. "*Managing technical debt with the sqale method*". *IEEE software*, (6), 44-51, 2012.

[15] W. Cunningham, The WyCash portfolio management system, ACM SIGPLANOOPS Messenger 4 (1992) 29–30.

[16] J.-L. Letouzey, "*The SQALE method for evaluating technical debt*", in: Proceedings of the Third International Workshop on Managing Technical Debt, Piscataway, NJ, USA, 2012, pp. 31–36.

[17] Available: http://www.sonarqube.org/

[18] A. Nugroho, J. Visser, T. Kuipers, "*An empirical model of technical debt and interest*", in: Proceedings of the 2nd Workshop on Managing Technical Debt, New York, NY, USA, 2011, pp. 1–8.

[19] K. Schmid, "*A formal approach to technical debt decision making*", in: Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures, 2013, pp. 153–162.

[20] P. A. Quezada-Sarmiento, L. E. Enciso-Quispe, J. Garbajosa and H. Washizaki, "Curricular design based in bodies of knowledge: Engineering education for the innovation and the industry," *2016 SAI Computing Conference (SAI)*, London, 2016, pp. 843-849. doi: 10.1109/SAI.2016.7556077

[21] C. Calderon-Cordova, M. Guajala-Michay, R. Barba-Guaman, and P. Quezada-Sarmiento, "Design of a machine vision applied to educational board game," in 2016 6th International Workshop on Computer Science and Engineering, WCSE 2016, 2016.

[22] P. A. Quezada-Sarmiento, L. Enciso and J. Garbajosa, "Use of body knowledge and cloud computing tools to develop software projects based in innovation," *2016 IEEE Global Engineering Education Conference (EDUCON)*, Abu Dhabi, 2016, pp. 267-272. doi: 10.1109/EDUCON.2016.7474564

[23] P. A. Quezada-Sarmiento, M. M. Quezada, L. Pacheco-Jara and J. Garbajosa, "Evaluation of occupational and professional profiles in Ecuadorian context based on guide of Knowledge SWEBOK and ontological model," *2016 Third International Conference on eDemocracy & eGovernment (ICEDEG)*, Sangolqui, 2016, pp. 42-47. doi: 10.1109/ICEDEG.2016.7461694