SindhUniv. Res. Jour. (Sci. Ser.) Vol. 51 (2) 327-332 (2019)

http://doi.org/10.26692/sujo/2019.6.53

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Identifying Code Quality issues in Student Projects

G. LAGHARI[++], K. DAHRI[*], S. NIZAMANI[**] M. Y. KOONDHAR[***], M. HYDER[***], A. H. ABRO[+],

Institute of Mathematics and Computer Science, University of Sindh, Jamshoro, Pakistan

Received 4th August 2018 and Revised 29st January 2019

Abstract: Software code quality is important specifically for the maintenance of the software. However, owing to many factors professional software developers accumulate a large amount of technical debt in their code bases. Students are no exception and they also commit code quality issues in their code. In this paper, we perform an exploratory analysis of student projects to understand which code quality issues are prevalent and more frequent. We find that almost all code quality issues are found in student projects. However, *Code Style*, *Documentation*, and *Design* are more frequently found than other issues.

Keywords: Student projects, software code quality, code smells.

1. INTRODUCTION

The software has become an essential and indispensable element in modern society. Contemporary society cannot function without software. Thus, software development is treated professionally like other professional disciplines under the umbrella of software engineering. The deriving element in software is the source code written to produce executable software.

The source code is, therefore, a valuable asset and needs to be managed very carefully. However, in the age of software development agility owing to many certain factors such as schedule or budget, the design choices and appropriate solutions are often neglected. This leads to the messy legacy code affecting the maintainability. Cunningham defined this phenomenon as *technical debt* (Cunningham 1993).

Similarly, Kent Beck refers to these code quality issues, which can be refactored to support the maintainability, as *code smells*(Martin 1999).After the inception of the term, code smells many best practices, code conventions, etc., and the tools to detect them have emerged. Code smells are one of the critical factors accumulating technical debt (Kruchten 2012).

Research studies on the code analysis of professional software developers indicate that industrial code suffers from the code quality issues. Many studies have been done to understand how these code smells are introduced in the code base. Some common causes include lack of skill or awareness, frequently changing requirements, development technology constraints, software processes, or schedule pressure, etc. (Sharma 2018).

Since in the university courses, the main focus gravitates towards teaching the students about the fundamental theories such as programming fundamentals, algorithms, and data structures, the main focus in the code is to get it working irrespective of the code quality (Dietz 2018). Moreover, students are also gradedby correctness in their solutions than the variety of those solutions. Once the assignments are completed, the code students produce is tossed into the trash. Thus, students lack incentives to improve the quality of their code, and their efforts surround on delivering the working solutions (Dietz 2018).

In this paper, we also posit on the fact that students' code heavily suffers from the code quality issues. We take the exploratory data analysis approach to verify our assumption and also understand which types of code quality issues are prevalent and how frequently.

The contributions of this paper are empirical evidence on the code quality issues in students' projects, which has the implications for the programming mentors to also teach clean code besides providing the students with fundamental theories.

In the rest of the paper, we provide the research design in Section 2 and report the results in Section 3. In Section 4, we provide some related work on the analysis of software code quality. Finally, we conclude the paper in Section 5.

2. RESEARCH DESIGN

To explore the code quality issues in student projects, we perform the exploratory data analysis. Bellow, we describe the datasets, analysis tools and methods, and the research questions we intend to answer.

[++]Corresponding Author: gulsher.laghari@usindh.edu.pk.
[*]Institute of Information and Communication Technology, University of Sindh, Jamshoro, Pakistan.
[**]Department of Information Technology, Sindh University Campus Mirpurkhas, Pakistan.
[***] Information Technology Centre, Sindh Agriculture University, Tandojam, Pakistan.
[+]Sindh University Laar Campus @ Badin, Pakistan

Dataset. Our dataset consists of final year projects of undergraduate students. We analyze eight final year projects whose source code is available. One of the projects is a java desktop application, whereas the remaining seven projects are Android apps. The code statistics of the projects are provided in **(Table 1)**. We use the cloctool (https://github.com/AlDanial/cloc), which is a static code analysis tool, to compute the code statistics. We can observe that our dataset contains projects from small of about 400 lines of code to large about 8000 lines of code.

Table 1. Code details of final year undergraduate student projects. The first project is a desktop Java application, and the remaining projects are Android apps.

Project	Java Files	Number of Lines		
		B	C	LOC
Code Generator From UML Class Diagram	8	819	176	2183
GSM Based Home Appliance Controlling System Using Android	9	207	47	440
NotificationSystem	26	755	238	1858
Online EasyBuy	22	695	7062	3529
OrderSystem	14	646	4348	3374
Remote Desktop Sharing of Computers on Android Phone	20	839	4635	4029
Secure Result Submission And Inquiry System	17	917	9142	8308
Wheelchair Application	10	590	5243	3598
TOTAL	**126**	**5468**	**30891**	**27319**

B = Blank Lines, C = Comments, LOC = Lines of Code

Code quality issues and tool. For the analysis of code quality issues, we employ a static source code analyzer tool called PMD (https://pmd.github.io/).

PMD detects common programming flaws by static analysis of the code and provides support to check many code quality rules.

In this paper, we check for all the following set of rules, which the PMD can easily detect if they are violated in the code.

Best Practices. These are the rules related to generally accepted best practices. For example, instead of printing the stack trace, it is best to log it.

Code Style. These rules are about a specific style of coding for example naming conventions.

Design. These rules dictate the design of the project. For example, the "Law of Demeter" restricts objects to "only talk to friends" to reduce coupling.

Documentation. These rules relate to documentation of the code for example whether or not the comments are required for particular elements.

*Error-prone.*These rules concern about the broken or confusing constructs in the code. For example,instead of comparing with hard-coded literals in conditionals it is best to declare constants.

Multithreading. These are the rules about threaded execution. For example, method level synchronization can be problematic.

Performance. These rules deal with code performance. For example, it is efficient to use array methods to copy the data between the arrays than to iterating the arrays.

Research questions. In this paper, we assume that the students' code does have code quality issues. To better explore by exploratory data analysis, we ask the following research questions.

RQ1.*Which code quality issues are present in the projects?*

RQ2.*How are the code quality issues distributed across the projects?*

RQ3.*Which are the most common code quality issues in student projects?*

3. RESULTS

In this section, we provide the analysis of the code quality of student projects and answer the research questions outlined in the research design section.

Answer to RQ1. Code quality analysis indeed confirms our assumption that students' code does suffer from quality issues. Table 2 provides the summary of code quality issues in all projects, where we can see that the code suffers from all quality issues. Amongst all issues, code style and documentation issues occur most frequently.

Table 2. A number of quality issues in the projects.

Rule Set	Instances
Best Practices	265
Code Style	36339
Design	1049
Documentation	26604
Error-Prone	649
Multithreading	29
Performance	170
TOTAL	**65105**

(Tables 3-6) provide further information on which specific rule violations occur for the generic categories summarized in **(Table 1).**

Table 3. The type and frequency of rule violations of Best Practices.

Rule	Instances
Unused Imports	74
Avoid Print Stack Trace	44
System Println	26
Position Literals First In Case Insensitive Comparisons	25
Position Literals First In Comparisons	18
Unused Local Variable	13
ForLoop Can Be Foreach	10
One Declaration Per Line	10
Unused Private Field	7
Avoid Reassigning Parameters	6
J Unit A ssertions Should Include Message	6
Switch Stmts Should Have Default	6
Use Collection Is Empty	6
Loose Coupling	5
Replace Vector With List	3
Unused Private Method	3
Avoid Using Hard Coded IP	2
Use Varargs	1
TOTAL	**265**

Table 4. The type and frequency of rule violations of Code Style.

Rule	Instances
Field Naming Conventions	18980
Long Variable	15010
Local Variable Could Be Final	735
Method Argument Could Be Final	427
Class Naming Conventions	226
Short Variable	188
Comment Default Access Modifier	163
Default Package	160
Short Class Name	107
At Least One Constructor	102
Control Statement Braces	74
Use Diamond Operator	37
Only One Return	18
Confusing Ternary	13
Linguistic Naming	13
Method Naming Conventions	11
Useless Parentheses	11
Local Variable Naming Conventions	9
Useless Qualified This	9
Unnecessary Local Before Return	8
Premature Declaration	7
Use Under scores In Numeric Literals	7
No Package	5
Duplicate Imports	4
Identical Catch Branches	4
Unnecessary Constructor	4
CallSuper In Constructor	2
Unnecessary Return	2
Avoid Final Local Variable	1
Formal Parameter Naming Conventions	1
Unnecessary Fully Qualified Name	1
TOTAL	**36339**

Table 5. The type and frequency of rule violations of Design.

Rule	Instances
Law Of Demeter	854
Avoid Catching Generic Exception	54
Immutable Field	29
Excessive Class Length	22
Ncss Count	13
Cyclomatic Complexity	12
Excessive Method Length	11
N Path Complexity	8
Data Class	7
Signature Declare Throws Exception	7
Singular Field	6
Simplify Boolean Expressions	5
Collapsible If Statements	4
Use Utility Class	4
Avoid Deeply Nested If Stmts	2
Excessive Imports	2
Simplify Boolean Returns	2
Too Many Methods	2
Use Object For Clearer API	2
Useless Overriding Method	2
Too Many Fields	1
TOTAL	**1049**

Table 6. The type and frequency of rule violations of Documentation.

Rule	Instances
Comment Required	16429
Comment Size	10167
Uncommented Empty Constructor	5
Uncommented Empty Method Body	3
TOTAL	**26604**

Table 7. The type and frequency of rule violations of Error-Prone.

Rule	Instances
Dataflow Anomaly Analysis	282
Bean Members Should Serialize	266
Avoid Duplicate Literals	27
Avoid Literals In If Condition	18
Import From Same Package	7
Empty Catch Block	6
Null Assignment	6
Use Equals To Compare Strings	6
Avoid Field Name Matching Method Name	4
Call Super Last	4
Do Not Call System Exit	4
Assignment In Operand	3
Test Class Without Test Cases	3
Use Locale With Case Conversions	3
Assignment To Non Final Static	2
Compare Objects With Equals	2
Empty If Stmt	2
Missing Serial Version UID	2
Avoid Catching NPE	1
Call Super First	1
TOTAL	649

Table 8. The type and frequency of rule violations of Multithreading.

Rule	Instances
Do Not Use Threads	28
Avoid Synchronized At Method Level	1
TOTAL	**29**

Table 9. The type and frequency of rule violations of Performance.

Rule	Instances
Add Empty String	48
Use String Buffer For String Appends	42
Avoid Instantiating Objects In Loops	33
Redundant Field Initializer	22
Avoid File Stream	8
String Instantiation	7
Use Array List Instead Of Vector	3
Inefficient String Buffering	2
Too Few Branches For A Switch Statement	2
Boolean Instantiation	1
Inefficient Empty String Check	1
Integer Instantiation	1
TOTAL	**170**

Answer to RQ2. The distribution of the code quality issues in the projects is depicted in **(Fig 1)**, while **(Fig 2)** provides the summary of these distributions. There we observe that five projects including the Notification System, Order System, Remote Desktop Sharing, Result Submission and Inquiry, and Wheel Chair application do have all types of code quality issues. It was observed that only three projects do not have all types of issues. Code Generator lacks Multithreading related issues, Home Appliance Controlling requires Performance related issues, and the Online EasyBuy lacks Error-Prone, Multithreading and Performance issues. We also notice that Code Style is the dominant code quality issue and that too in all projects. Note that the file count in Fig 1 and 2 does not mean unique files. Although there are only 8 files in Code Generator project yet, the figures show the file count for Code Style amounting to 50.This is due to the fact the there are many rules which co-occur in the same files. Thus the same file counts more than once.

Answer to RQ3. To answer this question, we report the topmost code quality issues that affect more files. Fig 3 provides the distribution of code quality issues that span in at least 10files.We observe that excepting Multithreading other issues are most common across projects, yet some issuesare only project specific. Thus, the most common issues are of type *Documentation*, more precisely the Comment Required affecting 109 files in 6 projects.

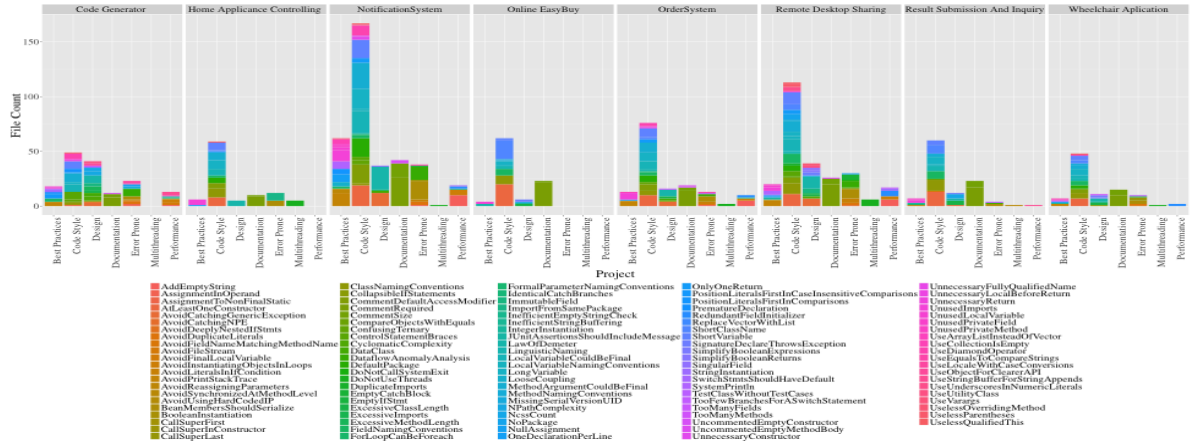

Fig. 1 The distribution of code quality issues in the projects.

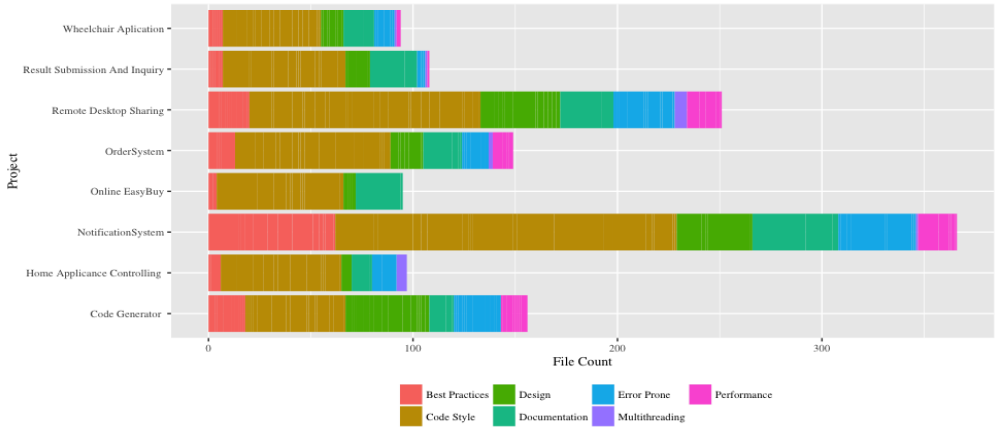

Fig. 2 The summary of distribution of code quality issues in the projects.

Similarly, the At Least One Constructor issue of *Code Style* affects 73 files across 5 projects. Other most common issues of *Code Style* in clude Method Argument Could Be Final with file count of 34, Local Variable Could Be Final with file count of 31, and Short Variable with file count of 27. On the other hand, Law of Demeter with file count of 32 is the second most common issue of *Documentation*.

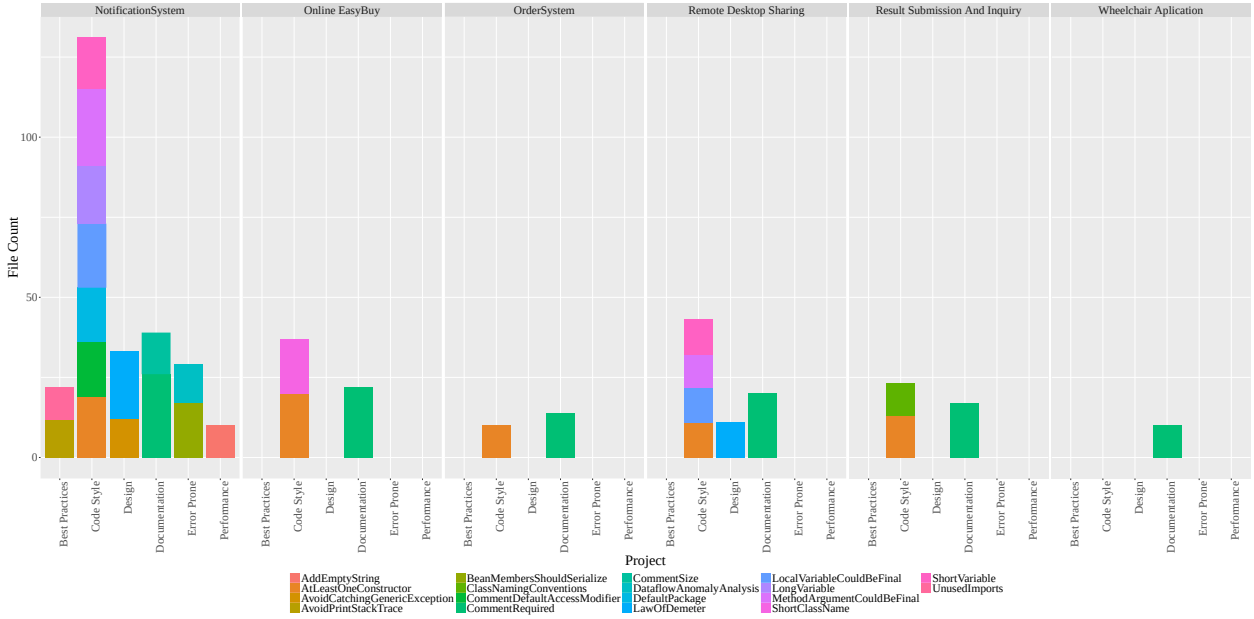

Fig. 3. Code quality issues which are distributed in atleast 10 files.

Discussion. With an exploratory analysis of a few student projects, we identify that the students' code substantially suffers from code quality issues and that almost all types of issuesare found in the code. More specifically students code lacks specific coding style and documentation, which suggests that students are more concerned in the working of their code than the quality of the code. More specifically, they pay less attention to the maintenance, since the code style and documentation are more helpful in maintenance.

Since our dataset is small and all the projects are written in Java, our results cannot be generalized. These findings provide surface hints on students' code quality. This can be helpful for the instructors to pay attention to code quality in students' code.

4. RELATED WORK

There are many studies on the code quality in the projects from kids (Aivaloglou 2016), university students (Altadmri 2015, Keuning 2017), and even professional developers (Monden 2002).

Aivaloglou and Felienne performed a large-scale study on Scratch projects. Scratch is a block level programming language primarily used to teach programming to kids. They observed that procedures and conditional loops were not commonly used in those projects and those projects had code smells more especially dead code and code clones (Aivaloglou 2016).

Keuning *et al.* 2017) explored code quality issues in large code bases of students and found that laymen student developers committed substantial code quality issues. Despite the use of the code quality tools, it had minimal effect on the improved code quality.

(Monden *et al.*, 2002). study the code clone, which is a duplicated code section, in a legacy software consisting of about one million lines of code. They quantitatively clarified the relation between code clones and the reliability and maintainability of the software. More specifically they identified that the modules with code clones expanding to 200 lines are less reliable and maintainable (Monden 2002).

(Dietz, *et al.*, 2018) recommend teaching clean code in the university. Inspired by the fact that universities can learn from industry to improve programming courses, they presented code review-driven course for undergraduates which uses static code analysis tools coupled with a book on code quality (Dietz 2018).

Stegeman created an empirically validated model of code quality, which applies to early programming courses (Stegeman 2014). Later they proposed a systematic grading scheme by designing a rubric that

uses their model to provide feedback to students (Stegeman 2016).

5. CONCLUSIONS

Developing software professionally is essential since modern society heavily relies upon software. Code smells are serious threats that contribute to the technical debt in software.

Since university courses mainly target teaching students about fundamental theories, students remain oblivious to these code quality issues until they join the industry.

In this paper, we performed the exploratory analysis of student code focusing on code quality and verified that students' code indeed accumulates huge technical debt. Our results indicate that almost all types of code quality issues occur in student projects. There are some issues that arise more frequently and are densely distributed in many files and projects. While some issues are less common. We found that *Code Style*, *Documentation*, and *Design* issues are more prevalent and frequent in student projects.

The results of this study can be helpful specially for the instructors and students to pay attention to code quality. This would result in students adopting software engineering principles early on. Thus, there are long-term implications, when students start working professionally.

REFERENCES:

Aivaloglou, E., and F. Hermans. (2016) "How kids code and how we know: An exploratory study on the Scratch repository." In Proceedings of the 2016 ACM Conference on International Computing, Research, 53-61. ACM.

Altadmri, A. and N. C. C Brown. (2015) "37 million compilations: Investigating novice programming mistakes in large-scale student data." In Proceedings of the 46th ACM Technical Symposium on Computer Science, 522-527. ACM.

Cunningham, W. (1993) "The Wy Cash portfolio management system" ACM SIGPLAN OOPS Messenger 4, no. 2: 29-30.

Dietz, L. W., J. Manner, S. Harrer, and J. Lenhard. (2018) "Teaching Clean Code." In Proceedings of the 1st Workshop on Innovative Software Engineering.

Keuning, H., B. Heeren, and J. Jeuring. (2017) "Code quality issues in student programs." In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science, 110-115. ACM.

Kruchten, P., L. R. Nord, and I. Ozkaya (2012). "Technical debt: From metaphor to theory and practice." IEEE software 29, no. 6: 18-21.

Martin, F., (1999) “Refactoring: improving the design of existing code”. Addison-Wesley

Monden, A., D. Nakae, T. Kamiya, Shin-ichi Sato, and Ken-ichi Matsumoto. (2002)."Software quality analysis by code clones in industrial legacy software." In Proceedings Eighth IEEE Symposium on Software Metrics, pp. 87-94. IEEE, Sharma, Tushar, and Diomidis Spinellis. "A survey on software smells." Journal of Systems and Software 138 (2018): 158-173.

Stegeman, M., E. Barendsen, and S. Smetsers. (2014) "Towards an empirically validated model for assessment of code quality." In Proceedings of the 14th Koli Calling international conference on computing research, 99-108. ACM.

Stegeman, M., E. Barendsen and S. Smetsers. (2016) "Designing a rubric for feedback on code quality in Programming Courses." In Proceedings of the 16th Koli Calling International Conference on Computing Research, 160-164. ACM.