PA2557 – QUALITY CONTROL

ASSIGNMENT 2

SRI SAI RIPUGHNA RISHITOSH SONTI
Department of Software Engineering
P-no: 19960916-8357
sosr17@student.bth.se

GANNA ANIL
Department of Software Engineering
P-no: 19960710-7514
gaam17@student.bth.se

MANPREETH BOLLU
Department of Software Engineering
P-no: 19970403-1237
bomn17@student.bth.se

SRAVANI BEJAWADA
Department of Software Engineering
P-no: 19970525-9563
besr@student.bth.se

PAVAN KUMAR PULA
Department of Software Engineering
P-no:19970506-6919
pupa17@student.bth.se

***Abstract* – This paper mainly focuses on evaluating the code quality through code reviews for a better insight into the maintainability, readability and understandability of the software. The code quality is evaluated using various automated tools and through manual code review as well. The comparison, results of the code reviews through the tools and manual evaluation help in improving the subjective of the code quality.**

INTRODUCTION:

Code quality is tested with the help of code reviews which make the code easy to understand, readable and maintainable. These code reviews can be implemented through some of the automation tools which are available online and manual evaluation can be carried out. In this assignment, we have made a search for an open source software as per the given limitations. We found an open source software which contains 20 classes and a checklist has been prepared. The checklist is made by performing a search on the code review checklists on the internet and some aspects were considered. Based on these aspects the open source code is evaluated and the code smells are identified. We have also evaluated the code through the automated tools and manually, then the results from both types of evaluation methods are compared.

SECTION – I

In the GitHub explore there were abundant open source software repositories in which we have chosen the code from an open source domain of the Chinese-Checkers, coded in Java platform.

Link to the Open Source Code Repository:

https://github.com/MarcelDudek/Chinese-Checkers

This source code contains two files in the source folder. The main folder contains 15 .java classes and the text folder contain 12 .java classes that are used as software UI. This source code was the latest working software version which was common and easy to understand if the game is known.

SECTION – II

We have selected three tools for our automated code reviews according to our checklists which were widely used and were free to use without having any license cost. In order to know the exact time at which there is a movement and investigate the code, the third tool is considered as the other tools were not successful in executing that kind of data.

The tools which are considered for our code analysis review were as follows:

- **Codacy**: Codacy is an open, extensible platform tool for analysis of the code i.e., inspects the health of the code. The tool gives an overview of all the code smells and code review. The tools execute are the test coverage, complexity, duplication, style of code. It also gives a maintainability check of the code whether the code is easy or difficult to maintain. The tool also gives an overview of the total source code in the form of a graph to have a clear description of the level of duplication, complexity and the unused variables.

- **CodeBeat**: It is an analysis tool which gathers the results of the code analyzed into a report and gives the information about the code maintainability percentage in the form of GPA. This tool is a real time analysis tool as it is free to use with no license cost and takes minimum 1 day to analyze the code and gives the complexity and duplication in the code. The rating and reviews of this tool are quite positive from the users. The tool also finds the unused variables instances, objects creations.

- **IntelliJ IDEA**: This is a Java integrated development environment for developing computer software. This tool also works as a static code analyzer tool which mainly focuses on the duplication in the code. This identifies the duplication such as copy instances, same variables, same Boolean variables, unused variables. This tool is freely available in the JetBrains repository with no license cost.

SECTION - III

Based on the previous assignment studies and available checklists online, the most common code smells from all the tools and checklists the following code smells are important for reviewing the code of an open source software:

Complexity:

The complexity of the code is defined as the level of complex code which is difficult to understand and difficult to maintain. The code smells that are to be inspected are separated into two types of complexity i.e., cognitive complexity & cyclomatic complexity. The cyclomatic complexity indicates the complexity in the code and gives a quantitative measure to solve the complexity of a program. The cognitive complexity is a measure of how difficult the code to understand and test [1].

- Are there any classes which have more cyclomatic complexity?
- Are there any classes which have more cognitive complexity?

Duplication:

The duplicate of the code is the more difficult part of the code where an instance of the code is repeated more than twice. The duplication of the code can be easily removed by explaining the code in one class or can the instance of the code can be redefined. Duplication is considered to be one of the main reasons for an increase in the level of maintainability of the code [2].

- Does the code contain any duplication scenarios that can be removed?

Styling:

The styling in the code is defined as the variables in the code which are written inappropriately, such as upper-case letters instead of lower-case letters, unused variables, lines spaces, non-comments etc. These type of styling errors may lead to code smells and the level of understandability and readability will be significantly more[3].

- Does the code include long methods with too many capital letters and more spacing?
- Are there any unused variables in the code which make it difficult to understand?

SECTION – IV

Evaluation of the open source software is accomplished by using the considered tools and the following are the values obtained related to our code review checklist.

Cyclomatic Complexity & Duplication by Codacy:

Classes	Complexity	Duplication
tp/chinesecheckers/klient/Plansza.java	9	9
tp/chinesecheckers/klient/Klient2. java	9	13
tp/chinesecheckers/klient/Piony.java	12	29
tp/chinesecheckers/serwer/SerwerGui.java	3	0
tp/chinesecheckers/klient/Klient.java	3	2
tp/chinesecheckers/klient/Pionytest.java	0	0
tp/chinesecheckers/Gradomyslnaserwer.java	98	4
tp/chinesecheckers/klient/Javatest2plansza.java	1	0
tp/chinesecheckers/klient/Ramkaplanszyodpaltest.java	1	0
tp/chinesecheckers/serwer/Serwer.java	14	0
tp/chinesecheckers/GraDomysInaSerwerTest.java	11	6
tp/chinesecheckers/klient/RamkaPlanszy.java	1	0
tp/chinesecheckers/serwer/Polaczenie.java	5	0
tp/chinesecheckers/exception/NiepoprawnaWiadomosc.java	1	0
tp/chinesecheckers/Gra.java	1	0
tp/chinesecheckers/Zawodnik.java	5	0
tp/chinesecheckers/TworcaGryDomyslnej.java	19	0
tp/chinesecheckers/exception/NiepoprawnyRuch.java	1	0
tp/chinesecheckers/TworcaGracza.java	7	1
tp/chinesecheckers/GraDomyslna.java	4	0
tp/chinesecheckers/Rozgrywkaserwer.java	0	0
tp/chinesecheckers/Tworcazawodnika.java	1	0
tp/chinesecheckers/Tworcagry.java	0	0
tp/chinesecheckers/Gradomyslnatest.java	1	0
tp/chinesecheckers/exception/NiepoprawnaWiadomoscTest.java	1	0
tp/chinesecheckers/klient/KlientTest.java	1	0
tp/chinesecheckers/serwer/SerwerTest.java	3	0
tp/chinesecheckers/klient/PlanszaTest.java	1	0
tp/chinesecheckers/exception/PionekNaTejPozycjiNieIstniejeTest.java	1	0
tp/chinesecheckers/klient/RamkaPlanszyOdpal.java	1	0
tp/chinesecheckers/TworcaGryDomyslnejSerwer.java	4	0
tp/chinesecheckers/TworcaGryDomyslnejTest.java	2	0
tp/chinesecheckers/Bot.java	1	0
tp/chinesecheckers/Pionek.java	1	0
/tp/chinesecheckers/exception/NiepoprawnyRuchTest.java	1	0
tp/chinesecheckers/GraDomyslna.java	4	0
tp/chinesecheckers/TworcaGracza.java	7	1
tp/chinesecheckers/Gracz.java	2	0
tp/chinesecheckers/PionekTest.java	1	0
tp/chinesecheckers/TworcaBota.java	7	1

exception/PionekNaTejPozycjiNieIstnieje.java	1	0
/klient/RamkaPlanszyTest.java	1	0
chinesecheckers/BotTest.java	1	0
chinesecheckers/TworcaBotaTest.java	1	0
tp/chinesecheckers/GraczTest.java	1	0
exception/PionekNaTejPozycjiNieIstnieje.java	1	0
TOTAL	241	66
Average	5.23	1.43

Cyclomatic Complexity by CodeBeat:

Classes	Complexity	Duplication
tp. chinesecheckers Klient.klient	36	1
tp. chinesecheckers Serwer.serwerGUI	30	0
tp. chinesecheckers Klient.Piony	18	30
Chinskie.Warcaby.Piony	34	5
Chinskie.Warcaby.Gracz	7	0
Chinskie.Warcaby.Plansza	38	0
tp. chinesecheckers. Klient.Odbiorcakomunikatow	3	1
tp. Chinesecheckers.Rozgrywkaserwer	0	0
Chinskie.Warcaby.serwer	21	0
tp. chinesecheckers. TworcaGryDomyslnejserwer	19	0
tp. chinesecheckers. Klient.RamkaPlanszyOdpal	3	0
Chinskie .Warcaby.RamkaPlanszyOdpal	3	0
Chinskie .Warcaby.ObrazRamka	5	0
tp. chinesecheckers. Klient.RamkaPlanszy	5	0
Chinskie .Warcaby.serwer.Obslugaklientow	0	0
tp. chinesecheckers. Serwer.AkutalizatorListyGraczy	4	0
tp. chinesecheckers. Gracz	4	0
Chinskie.exception. NiepoprawnyRuch	0	0
Chinskie .Warcaby .Gracz.PrzyciskwyslijListener	6	0
Chinskie .Warcaby.Obrazektest	1	0
tp. chinesecheckers. Klient.klient. PrzyciskwyslijListener	6	0
Chinskie .Warcaby.testObrazka	3	0
Chinskie .Warcaby. Gracz.MouseTestPanel	5	0
Chinskie.Warcaby.RamkaPlanszy	5	0
Chinskie .Warcaby. Gracz. Odbiorcakomunikatow	0	0
Chinskie.Warcaby. Gracz.Frame	1	0
tp. chinesecheckers. Bot	0	0
TOTAL	148	37
Average	5.48	1.37

Cognitive Compexity by Codacy:

Classes	Complexity
tp/chinesecheckers/klient/Piony.java	5
exception/PionekNaTejPozycjiNieIstnieje.java	1
TOTAL	6

Average	3

Duplication Code by IntelliJ IDEA:

In the IntelliJ IDEA, the duplication of the code is shown. The lines of code where there is duplication i.e., the same code has been reused without any change in the variables and presence of unused variables. This can also be defined as the code was repeated more than twice in several classes. By this, the level of maintainability of the code will increase. Moreover, duplication makes the code more difficult to understand.

Classes	Positions of Duplication of Code
tp/chinesecheckers/klient/Piony.java	Lines 565 to 606 & Lines 524 to 556 Lines 73 to 117 & Lines 60 to 103 Lines 608 to 643 & Lines 558 to 585 Lines 120 to 155 & Lines 106 to 141 Lines 17 to 52 & Lines 15 to 50 Lines 311 to 346 & Lines 240 to 275 Lines 238 to 273 & Lines 195 to 230 Lines 180 to 215 & Lines 151 to 186 Lines 537 to 564 & Lines 500 to 523 Lines 155 to 176 & Lines 513 to 534 Lines 52 to 69 & Lines 469 to 486 Lines 490 to 505 & Lines 463 to 475
tp/chinesecheckers/TworcaBotaTest.java tp/chinesecheckers/TworcaGracza.java	Lines 72 to 83 in TworcaBota & Lines 77 to 88 in TworcaGracza Lines 90 to 101 TworcaBota & Lines 95 to 106 in TworcaGracza Lines 41 to 54 in TworcaBota & Lines 48 to 61 in TworcaGracza Lines 109 to 160 in TworcaBota & Lines 114 to 165 in TworcaGracza
tp/chinesecheckers/GraDomysInaSerwerTest.java	Lines 15 to 31 & Lines 78 to 94 Lines 102 and 103
tp/chinesecheckers/klient/Klient2. java	Lines 260 to 274 & Lines 280 to 294 Lines 188 to 221 in klient2 & Lines 172 to 204 in plansza Lines 323 to 328 in Klient2 & Lines 347 to 353 in Plansza
tp/chinesecheckers/klient/Plansza.java	Lines 300 to 310 & Lines 315 to 325 Lines 225 to 237 in klient2 & Lines 208 to 222 in plansza Lines 298 to 317 in klient2 & Lines 329 to 342 in plansza Lines 42 to 59 in obrazek & Lines 33 to 50 in plansza Lines 111 to 131 in klient2 & Lines 92 to 111 in plansza

	Lines 76 to 88 in klient & Lines 414 to 426 in klient2 Lines 660 to 672 in klient & Lines 127 to 142 in gracz Lines 243 to 250 in klient2 & Lines 284 to 291 in Plansza & Lines 111 to 118 in Plansza Lines 331 to 333 in klient2 & Lines 362 to 372 in plansza
tp/chinesecheckers/Zawodnik.java	Lines 131 to 135 & Lines 153 to 157 Lines 55 to 58 & Lines 62 to 65 Lines 90 to 104 in OdbiorcaKomunikatow & Lines 162 to 182
Chinskie .Warcaby.Obrazektest.java	Lines 183 to 190 & Lines 241 to 247 Lines 296 to 303 & Lines 353 to 359 Lines 70 to 76 & Lines 127 to 133 Lines 522 to 531 in obrazek & Lines 77 to 81 in plansza
tp/chinesecheckers/GraDomyslnaSerwer.java	Lines 327 and 650 contents Lines 90 in Zawodnik and Lines 632 in GraDomyslnaSerwer
/klient/RamkaPlanszyTest.java	Lines 663 to 667 in RamkaPlanszy & Lines 76 to 79 in Frame Lines 21 to 25 in RamkaPlanszy & Lines 12 to 16 in ObrazRamkam & Lines 18 to 22 Ramkaplanszy
tp. chinesecheckers. Klient.klient. PrzyciskwyslijListener	Lines 64 to 69 in PrzyciskWyslijListener & Lines 395 to 403 & Lines 641 to 647 & Lines 148 to 156
tp/chinesecheckers/TworcaGryDomyslnejTest.java	Lines 44 & Lines 45 in TworcaGryDomyslnejTest Lines 39 & Lines 40 in TworcaGryDomyslnejTest

Code Style by Codacy:

The code smells occurred as the names of classes used in the code had upper-case letters which is a syntax error. The unused variables are in the code are high. Removal of these code smells will result in an increase in the level of readability and understandability of the code which in turn results in the decrease in the maintenance effort.

Classes	Capital Letters
tp/chinesecheckers/klient/Plansza.java	11
tp/chinesecheckers/klient/Klient2. java	1
tp/chinesecheckers/klient/Piony.java	7
tp/chinesecheckers/serwer/SerwerGui.java	2
tp/chinesecheckers/klient/Javatest2plansza.java	1

TOTAL	22
Average	4.4

Classes	Unused Variables
tp/chinesecheckers/klient/Plansza.java	2
tp/chinesecheckers/klient/Klient2. java	1
tp/chinesecheckers/serwer/SerwerGui.java	4
tp/chinesecheckers/klient/Pionytest.java	7
tp/chinesecheckers/Gradomyslnaserwer.java	1
tp/chinesecheckers/klient/Javatest2plansza.java	5
tp/chinesecheckers/klient/Ramkaplanszyodpaltest.java	4
tp/chinesecheckers/GraDomysInaSerwerTest.java	2
tp/chinesecheckers/klient/RamkaPlanszy.java	1
TOTAL	27
Average	3

Manual Review:

While evaluating the code manually there are several instances where the logic of the code was very difficult to understand. The complexity of the code was increasing as the bugs were frequently appearing. There are many classes in which the conditional statements were dependent, and they will be no chance of errorless code since one change in the conditional statements will affect the entire code. The conditional statements cannot be avoided and when compared to the current implementation it might affect the maintainability and understandability of the code. However, manual evaluation of the code might take a long period of time.

The cognitive complexity in the code is too high for some of the classes and the unit test of the code might be difficult to understand which results in an increase of the maintenance effort. If the developers want to reduce the complexity in the code for an independent class, they should investigate the dependent classes which are inherited from the parent classes and make the change the class definition as per the requirements.

About the duplication of the code, there are several instances where the code is repeated more than twice i.e., the code is duplicated. In the Chinese-checkers game, there would be many positions of the object similar to the opposition at the exact same instance. This is where the tool execution fails as the code which is used in the loop segment has several chances to repeat more than twice. The tool analyzes that the code instant variables are repeated and so as a result, it will be given as duplicated code in the automation tool whereas it is not the same in a manual review. This also occurs when there is the repetition of the lines of code more than once. The tools fail to differentiate between the code duplication and the code which is useful at that moment.

While evaluating the code manually the public classes were starting with the upper-case letters. This might increase the maintainability but can easily be modified by changing the letters. There are also some unused variables or characters which should be removed to easily understand the code and avoid code smells.

SECTION – V

Comparison Of Outcome Of The Tools:

By comparing the results obtained from all the tools, we determine that the tools haven’t covered all the aspects in the checklist.

For the Cyclomatic complexity, Codacy and CodeBeat were able to give the complexity values in which both values are almost similar. However, the IntelliJ IDEA has failed to give the results for cyclomatic complexity. The cyclomatic complexity in both the tools are high and thus increase the level of maintainability.

For Cognitive Complexity, only Codacy has given the values for complexity which is very high and results in increasing maintanbility. The CodeBeat and the IntelliJ IDEA didn’t give any results for the cognitive complexity.

For the duplicate code, all three tools have given values in which Coadacy and CodeBeat values resulted similarly. The IntelliJ IDEA has given the positions of the code where it has been repeated more than twice.

Lastly, for the code style, Codacy has given the values for the number of unused variables and capital letters which can be easily modified without any affect in maintainability. The CodeBeat and IntelliJ IDEA didn’t give results regarding the unused variables and capital letters.

The CodeBeat and IntelliJ IDEA has given the total number of lines of code where the code smells have occurred whereas the Codacy didn’t give any results about the lines of code.

SECTION – VI

Comparison Of The Manual Evaluation And Outcome Of The Tools:

The code review values of the automated tools and the manual code review do not always give the same results. This is because the tools give mere values regarding the complexity and duplication of the code through which we cannot determine the maintainability and understandability. The values may differ through the experience and expertise of the developers. All the tools do not give the same values of the code as some of the tools consider the classes which are long as classes with high complexity [2]. But when evaluated manually it will be easy to understand the code.

CodeBeat gives an overview of the estimation of the code smells and maintainability in the form of percentage in GPA. In the duplication of the code, at first manual evaluation we found that the maintainability might get affected due to several numbers of duplications but when analyzed by the tool there were many duplications which were not real[4]. So, the tools might raise some false notifications even if they were correct at that position. The effect of the cyclomatic complexity will be high as there were many classes where the values of cyclomatic complexity are very high. The cognitive complexity value is also correct in the Codacy tool as the class was very complex to understand. But in the manual evaluation, the cyclomatic complexity does not affect much of the code understandability and maintainability. The results of the manual evaluation and the automated tool evaluation of the code conclude that the values found for cognitive complexity are found to be almost similar i.e., the high values may affect the code maintainability, readability, and understandability [2].

The unused variables of the code and the capital letters on the public class which were found during both the manual inspection and tool inspection does not affect much of the code maintainability as the variables can be removed if not used and the upper-case letters can also be modified [3]. The classes which have a high chance of complexity are dependent on the inspections are done by an individual developer and this may be a key factor in determining the maintainability and understandability of the code. We found two classes namely, “main” and “test”, out of which the “main” class had a significantly higher complexity which would, in turn, affect the maintainability of the code.

REFERENCES:

[1] A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.

[2] C. F. Kemerer and M. C. Paulk, “The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data,” *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 534–550, Jul. 2009.

[3] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, “The influence of non-technical factors on code review,” in *2013 20th Working Conference on Reverse Engineering (WCRE)*, Koblenz, Germany, 2013, pp. 122–131.

[4] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, “The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK, and ITK projects,” in *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, Hyderabad, India, 2014, pp. 192–201.