Source Code Analysis of GitHub Projects from E-Commerce and Game Domains

Doga Babacan
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: dogababacan96@gmail.com

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

Abstract— **The nature of domains, such as e-commerce, affects the software development process and the resulting software. Various domains may have similarities and differences with respect to each other under source code analysis. This research project examines the similarities and differences between game and e-commerce domains. With the technology now available to everyone, finding and examining public repositories is more straightforward. The domains chosen for this project are game and e-commerce since they are two of the most popular topics. In this research, inspections are made on 25 projects, 15 from the e-commerce domain and ten from the game domain. Developing a repository mining program that works with a software analysis tool and returns the results of this analysis is also validated within this research.**

Keywords-static source code analysis; repository mining; e-commerce software; game software.

I. INTRODUCTION

Static source code analysis is a way to analyze the code without running it. Nowadays, many tools help software developers to perform this process. In the literature, research was not found that utilizes these tools to inspect multiple repositories simultaneously and compares the results depending on their similarities and differences. If automation like inspection is possible for various repositories with these kinds of tools, it may be used in many types of research for many reasons. The SonarQube is utilized for this research. It measures technical debt, number of bugs, classes, functions, complexity, cognitive complexity, etc. These values may be used in many ways and inspected for relations between them. With these values in our hands, domains in software development, like e-commerce and game, can be studied, focusing on how they behave according to the results, whether they act similarly or not.

The main objective of this research project is to find out if automation applies to these kinds of tools during research with software, which clones many projects and, analyzes them, retrieves the results. Doing sample research utilizing this software will be another task to do. Each value in the results will be another attribute to compare and inspect. The sample research looks at the behavior of game and e-commerce domains, considering their results from the source code analyses tool. Each domain will be examined separately, and there will be a comparison. Public repositories of GitHub will be used for this purpose since it is one of the most popular code-storing and managing platforms.

The proposed solution uses Python language to create software that clones repositories from each domain, namely game, and e-commerce, to local with the get requests and python library for GitHub and upload them to SonarQube by utilizing the Python package SonarQube Client to analyze those repositories. After analyzing the repositories with SonarQube, the proposed solution continues by getting each project source code analysis result with the SonarQube Client package, inspecting those results with correlation matrices for each domain, and choosing specific attributes to examine the relation between them depending on the correlation matrix.

Java projects from GitHub in the e-commerce and game domains are the focus of this research. Some of the projects cannot be analyzed by SonarQube and they are excluded from research. Projects with other programming languages from the same domains will be considered in the future.

The paper is organized as follows: Section II presents the related work. Section III explains the proposed approach. Section IV presents the result and discussion, and the last section concludes the paper.

II. RELATED WORK

Sokol et al. [1] researched software mining tools, how they work, and the alternatives for this type of program. Research mainly focuses on Metric Miner's results and adds some points on Sonar.

Spadini et al. [2] developed a mining software repository program PyDriller, using Python language and put it against Python Framework GitPython. With fewer lines of code and less complexity, the results of both programs are compared.

Dabic et al. [3] developed another mining software for GitHub projects named GitHub Search. This program works in ten languages. It is a dataset that contains information about more than 700.000 public repositories in GitHub.

Dueñas et al. [4] introduced GrimoireLab, an open-source set of Python tools used in repository mining, analyzing, and visualizing. Third parties can also use the tool, designed as a modular toolset.

Koetter et al. [5] utilized SourceMeter to calculate chosen student project metrics. For each project, a Python tool developed by the article's authors was used for the benchmark calculation. With the gathered results, they made comparisons.

 ISBN: 978-1-68558-098-8

Jarczyk et al. [6] worked on determining two metrics that indicate the quality of GitHub projects. The initial statistic is derived from the ratings assigned to a project by other members of GitHub, while the second metric is derived through the application of survival analysis techniques to issues reported by users of the project. Following the development of the metrics, they proceeded to collect data on various attributes of many GitHub projects. Subsequently, they conducted an analysis utilizing statistical regression techniques to examine the impact of these attributes on the overall quality of the projects.

Yalçın and Tuglular [7] worked on 21 projects from GitHub with multiple versions of a tool the author created. JSoup and Selenium are utilized in the mining process. For each project, the author looked at whether the executable and test codes are increasing in sync, whether updates affect the co-evolution of test and executable data. In using GitHub software projects from different angle, AlMarzouq et al. [8] highlighted the challenges and opportunities of using GitHub as a data source in both research and programming education.

Gousios and Spinellis [9] found that the acquisition of data from GitHub is not a straightforward task, the suitability of the data for various research purposes may be limited, and the misuse of this data can potentially result in biased outcomes. Our findings match with their findings.

III. PROPOSED APPROACH

The proposed approach is composed of three steps:

1. Data Collection from GitHub
2. Source Code Analysis using SonarQube
3. Data preparation
4. Data analysis

The first three steps are explained in detail in this section. The fourth step is presented in the following section with results and discussion.

A. Data Collection from GitHub

The primary way of searching for software projects in GitHub is performed with a get request, through Python, such as "https://api.github.com/search/repositories?q=e-commerceis:featured+language:java&sort=stars&order=desc &per_page=100&page=1". The "is: featured" part of the string helps for searching topics in GitHub. If this part is not used, the result will return as a general search instead of topics. "language" filters for the asked language. "sort" lets the user choose which attribute to sort. In this research, the number of project stars is focused on finding a more reliable project on GitHub. "per_page" is the number of projects to be returned on request.

We intend to inspect the code metrics such as code smells, bugs, security hotspots, duplications, etc. We write a code that clones each release of a GitHub project and lists them as files in a folder if it did not have a release history to download; the code looks into previous tags of the project in GitHub, if it had tags, program clones each tag's repository and list as each of the version with its project name and its tag next to it. Also, it creates each version's SonarQube project under SonarQube.

By utilizing the "OS" library already included in Python, the directory for each repository can be created with a chosen name with "os. mkdir(path)". Here path is the whole path to the location, including the directory name such as "C:/Programming/RepositoryInspectionProject/3091E-c-o-Mshopizer".

When cloning from GitHub, the code "git clone {repo_url} {directory_name}" is written inside "os. system()" because it needs to be written as a console command. "repo_url" refers to the cloning URL of the repository, and "directory_name" is the name pattern that was chosen before as "3091E-c-o-Mshopizer". After cloning each project, there is a second step for them to upload these Maven projects to SonarQube for inspection. First, the creation of the project on the SonarQube is needed. This is performed through the utilization of the SonarQube Client library on Python. The package can be used by entering the username, password, and URL of the SonarQube installed on the computer. To create projects, the line "sonar.projects.create_project()" project name is placed as an argument where it is chosen as the directory name.

After creating the projects placed on SonarQube, repositories can be uploaded. This is achieved by utilizing the line, "mvn clean verify sonar : sonar -D maven.test.skip = true -D sonar.projectKey = {projectKey} -D sonar.host.url = http://localhost:9000 -D sonar. login = **********", here we skip tests by using "maven.test.skip = true" because tests could not be followed when trying automation on this research project.

B. Source Code Analysis using SonarQube

SonarQube is one of the best static source code analysis tools [10]–[12]. SonarQube is a Sonar Source product, and approximately seven million people utilize Sonar Source products currently [13]. SonarQube works with more than thirty languages, and one of them is Java.

The process for source code analysis starts when the cloning and uploading process is completed. To retrieve the results from SonarQube, SonarQube Client is utilized. Data for the following metrics [14] are collected:

Complexity: Complexity (cyclomatic complexity) is a metric where the number of paths in a code is calculated, and the minimum value of the function is 1. When the control flow of a piece of code diverges, the complexity increases. This calculation may differ depending on the language being used.

Cognitive Complexity: Cognitive complexity is a more detailed way of inspecting the complexity of a code. It is not a quantitative way of measuring as it is in cyclomatic complexity; it also counts in the degree of interconnectedness and abstraction or indirection in a piece of code. Cognitive complexity shows how understandable the code is and how much it is easy to maintain.

Issues: If any piece of code breaks the coding rule, it will be counted as an issue. There are three types of issues, which are bug, vulnerability, and code smell.

 ISBN: 978-1-68558-098-8

Violations: Any form of issue is also called a violation. Prefixes change depending on the importance of the violation; it can be blocker, critical, major, minor, and info.

Security Hotspots: A piece of code that is security sensitive; however, it is not as crucial as vulnerability; these hotspots may not impact the whole software, unlike the vulnerability.

Lines: Number of physical lines.

Lines of Code: The number of physical lines that contain at least one character. However, this character will not be counted if it is whitespace, tabular space, or part of a comment.

Functions: Number of functions.

Statements: Number of statements.

Comments: Number of comment lines in code.

Duplicated Lines: Number of duplicated lines in code.

C. Data Preparation

After source code analysis finished, then the data is normalized. The values for the metrics are placed in a dictionary and converted into a data frame to save as a CSV file, which are given in Table 1 and Table 3. By doing this, it becomes easier to work with the results on the Jupyter Notebook. On the Jupyter Notebook, after opening the CSV file, the data is converted to the data frame again to work on the values. All of the data (except star count and lines of code) is divided by a line of code because we want our values to be independent of the line of code of the repositories. Then, all the values are scaled to fit between 0 and 1. When the data preparation is finished, the correlation matrix is created to see the relationships among all attributes as shown in Table 2 and Table 4.

IV. Results and Discussion

The correlation matrices for the e-commerce and game domains are shown in Table 2 and Table 4. When we compare these two matrices, we see some differences between them. Positive and negative relations are different for game and e-commerce domains. The results are expected for the e-commerce domain; for instance, it is likely that with the decreasing number of classes, we expect a higher number of bugs which means there should be a negative relation between those two values. However, this does not apply to the game domain. This can be due to some outliers. The diagram lets the user see which attributes have positive and negative relations.

First, pair of attributes are selected. The first pair will be the number of comment lines and the number of code smells. It is a fact that code should explain itself without needing much of an explanation. These explanations are done with comment lines in the code. Code smells also tells us the software developer does not have much experience in writing code, most probably not following specific rules, does not apply tests, etc. A positive relationship is expected between them. The second pair is chosen as the number of bugs and the number of classes. If the number of classes increases, software may be thought to be cleaner and more organized and may be considered leading to fewer bugs.

When starting with the first pair of attributes, comment lines, and code smells, the correlation matrix in Table 2 shows a positive relation which was expected; the value of 0.61 is close to value 1, which means the relationship is strong even though it is not the strongest in the matrix. When a scatter graph is drawn, it shows each data point. There are outlier-like values on the diagram. To be sure, box plots are utilized. With the boxplots it is decided that two outliers need to be removed. After removing the outliers, the linear regression line is drawn in Figure 1 with (1). Also, the linear regression line shows us the positive relation better since the line has a visible positive slope.

$$y = 0.294\ x + 0.1 \qquad (1)$$

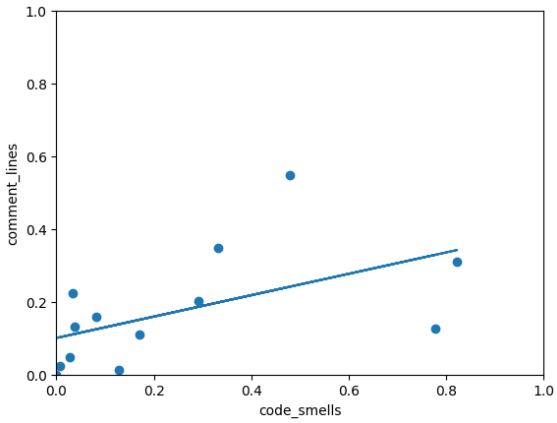

Figure 1. Linear Regression Line of Number of Comment Lines vs. Number of Code Smells of E-Commerce Domain.

The second pair of attributes, namely the number of bugs and the number of classes, are drawn on another scatter graph. Again, boxplots are utilized for each attribute to check the outliers, and it is verified that there are no outliers in this data set. The linear regression line is shown in Figure 2. The line has good visibility and a negative slope, showing a negative relation with (2).

$$y = -0.63\ x + 0.772 \qquad (2)$$

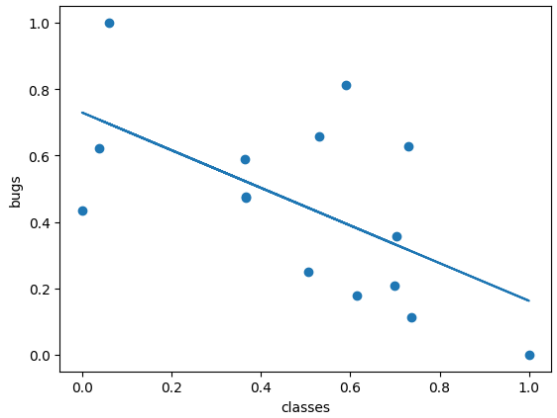

Figure 2. Linear Regression Number of Bugs vs. Number of Classes of E-Commerce Domain.

 ISBN: 978-1-68558-098-8

The first pair of attributes of the game domain are code smells and comment lines. Since there seems to be outliers on the scatter graph, so they are checked with the boxplots of each attribute. The boxplot showed that the two values with the number of comment lines value close to 1.0 are the outliers. After removing the outliers, a linear regression line is drawn in Figure 3 with (3).

$$y = 0.315 x - 0.025 \quad (3)$$

The slope can be seen on the graph as positive and the equation as positive. So, as expected, if the comment lines increase, more code smells can be expected in the software.

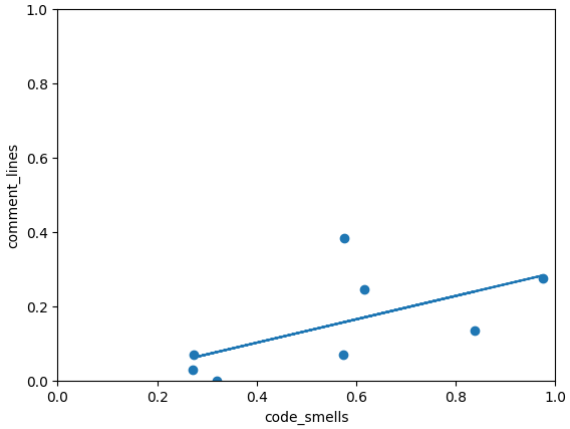

Figure 3. Linear Regression Line of Number of Comment Lines vs. Number of Code Smells of Game Domain.

The scatter graph shows how the data points are spread on the last pair of attributes set of game domain, number of classes, and number of bugs. When the outliers are checked with boxplots of each attribute, on each attribute, there is a different outlier; the number of outliers is decided as two. After removing the outliers, the scatter graph in Figure 4 is drawn with a linear regression line as in (4).

$$y = -2.586 x + 0.755 \quad (4)$$

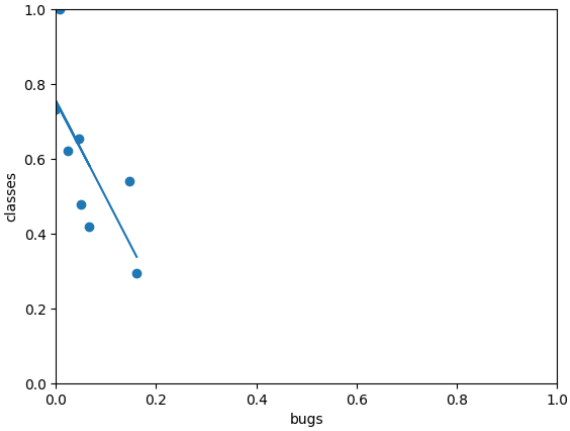

Figure 4. Linear Regression Number of Bugs vs. Number of Classes of Game Domain.

The negative slope can be seen on the graph and the equation, which means the relationship between the two attributes is negative.

There are concerns related to the generalization of results. First, all attributes should be interpreted relative to the local context; there are no absolute always correct interpretations. Second, although the projects are coded in Java, they are not necessarily object-oriented. Therefore, the results cannot be generalized to object-oriented projects. The generalizability of the research findings is limited both within the specific areas under investigation and to other domains for the following reasons. The research employs a limited sample size, and the findings lack sufficient statistical significance to generalize to the broader population. The study sample may lack representativeness in relation to the entire population. The research employs a non-random sampling technique, which has the potential to induce bias. The present study used a proprietary instrument devised by the researchers, which may potentially exhibit certain flaws or limits.

V. CONCLUSION

In this research, a software is developed to clone repositories and analyze them using SonarQube. Two domains, namely e-commerce and game domains, are analyzed. The correlation matrices showed that there is a difference between the two domains. The difference in the game domain can be due to structure and developers in general. However, in e-commerce, the developers follow specific rules and patterns while developing software which is common in software development. Two pairs of attributes from each domain are examined individually. The linear regression line is drawn, and the equation of the linear regression lines is shown. In conclusion, this project showed that automation could apply to repository mining, analyzing the source code, and retrieving the results of this analysis.

In this research, we first learned that the projects in GitHub are not necessarily well structured. Fetching the projects automatically was not simple and easy. Moreover, only some of the Java projects were analyzable by SonarQube. Therefore, we choose Java projects with Maven. Still, we cannot analyze all projects in the selected domains. Another source code analyzer may be used. A pluggable pipeline would be nice to have. We expected both domain projects we analyzed to be more fit to software engineering principles and best practices, but it wasn't the case.

For future work, we first plan to include more projects from the same domains and then perform cluster analysis to find the natural groups in the datasets, which can show trends, structures, or groupings that aren't obvious at first glance. This way, we plan to obtain useful insights for root causes and predictions. We also plan to figure out the dependencies between attributes.

The free version of SonarQube is employed in this research and it is limited. We would like to use the paid version for further analysis. We also plan to include other source code analysis tools such as ChatGPT and GitHub Copilot. Furthermore, some software engineering analysis tools will be included into the future research. They might

 ISBN: 978-1-68558-098-8

give us some perspectives from software engineering point of view, such as how many people did PRs on the same part of the source code, whether there are any correlations between the design patterns or technical debt and code quality, and whether code quality is related to the organizational structure of the project team.

Moreover, we plan to expand this research to include projects from the same domains with different programming languages as well as other domains, such as IoT, Healthcare, Sports.

REFERENCES

[1] F. Z. Sokol, M. F. Aniche, and M. A. Gerosa, "MetricMiner: Supporting researchers in mining software repositories," in 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM), Sep. 2013, pp. 142–146. doi: 10.1109/SCAM.2013.6648195.

[2] D. Spadini, M. Aniche, and A. Bacchelli, "PyDriller: Python framework for mining software repositories," in Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, in ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 908–911. doi: 10.1145/3236024.3264598.

[3] O. Dabic, E. Aghajani, and G. Bavota, "Sampling Projects in GitHub for MSR Studies," in 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), May 2021, pp. 560–564. doi: 10.1109/MSR52588.2021.00074.

[4] "GrimoireLab: A toolset for software development analytics [PeerJ]," Retrieved: July, 2023 [Online]. Available from: https://peerj.com/articles/cs-601/.

[5] F. Koetter, et al., "Assessing Software Quality of Agile Student Projects by Data-mining Software Repositories:," in Proceedings of the 11th International Conference on Computer Supported Education, Heraklion, Crete, Greece: SCITEPRESS - Science and Technology Publications, 2019, pp. 244–251. doi: 10.5220/0007688602440251.

[6] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki, "Github projects. quality analysis of open-source software," in Social Informatics: 6th International Conference, SocInfo 2014, Barcelona, Spain, November 11-13, pp. 80-94. Springer International Publishing, 2014.

[7] A. G. Yalçın and T. Tuglular, "Studying the Co-Evolution of Source Code and Acceptance Tests," Int. J. Softw. Eng. Knowl. Eng., pp. 1–27, Apr. 2023, doi: 10.1142/S0218194023500237.

[8] M. AlMarzouq, A. AlZaidan, and J. AlDallal, "Mining GitHub for research and education: challenges and opportunities," International Journal of Web Information Systems 2020, 16, no. 4, pp. 451-473.

[9] G. Gousios, and D. Spinellis, "Mining software engineering data from GitHub," in 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 501-502. IEEE, 2017.

[10] "TOP 40 Static Code Analysis Tools (Best Source Code Analysis Tools)," Retrieved: July, 2023 [Online]. Available from: https://www.softwaretestinghelp.com/tools/top-40-static-code-analysis-tools/.

[11] "Best Static Code Analysis Tools in 2023 | Compare Reviews on 90+ | G2," Retrieved: July, 2023 [Online]. Available from: https://www.g2.com/categories/static-code-analysis.

[12] L. Zelleke, "6 Best Static Code Analysis Tools for 2023 (Paid & Free)," Comparitech, Sep. 05, 2021. Retrieved: July, 2023 [Online]. Available from: https://www.comparitech.com/net-admin/best-static-code-analysis-tools/.

[13] "Clean Code Tools for Writing Clear, Readable & Understandable Secure Quality Code," Retrieved: July, 2023 [Online]. Available from: https://www.sonarsource.com/.

[14] "Metric definition," Retrieved: July, 2023 [Online]. Available from: https://docs.sonarsource.com/sonarqube/ latest/user-guide/metric-definitions/.

TABLE I. MINED DATA FROM GITHUB FOR E-COMMERCE DOMAIN

	blocker_violations	bugs	classes	code_smells	cognitive_complexity	comment_lines	complexity	critical_violations	duplicated_lines	file_complexity	functions	lines	major_violations	ncloc	security_hotspots	stars	statements	vulnerabilities
demo-microservices	0	2	70	19	8	24	80	3	0	1,1	114	3118	12	2641	1	29	144	0
double-shop	0	3	196	49	204	263	650	5	156	3,3	636	8363	26	6211	4	34	1246	0
eCommerce-JavaBackend	1	5	37	134	50	85	207	7	92	5,6	192	1942	80	1453	4	13	356	0
spring-restapi-ecommerce	1	7	58	57	75	52	360	10	230	6,3	320	3721	24	2517	9	42	687	2
e-commerce-database	1	0	47	20	2	3	71	2	0	1,5	57	1400	10	1110	1	15	111	0
ecommerce-microservice-backend-app	11	7	237	34	8	1	389	6	1172	1,7	508	12267	8	7906	1	84	597	0
eCommerceWebsite	6	143	153	711	1776	1628	2326	269	16441	7,9	1166	67448	318	54279	103	41	6788	209
eMusicStore_eCommerce_Website	3	2	13	89	30	151	98	8	0	7,5	86	1272	48	800	15	15	271	3
ShoppingCart	0	50	44	99	122	291	378	36	161	4,9	293	13059	83	11795	31	297	572	5
open-commerce-search	5	37	298	794	2925	3420	3349	101	362	13,1	1151	26892	173	18289	4	31	6528	1
SBootApiEcomMVCHibernate	10	8	166	267	291	287	1403	35	217	8,7	1245	10293	141	7534	13	15	2246	0
shopizer	45	146	1193	4049	6911	7579	10445	1145	6205	9	7606	110936	1636	73042	35	3091	25118	525
shopping-cart	11	55	37	150	216	138	314	32	464	5,6	220	34203	121	29869	89	56	1303	60
DevOps-E-commerce-	0	4	47	60	8	99	104	14	0	2,2	149	2073	26	1506	6	17	218	14
E-Commerce-Spring-Boot	1	2	40	116	101	32	182	4	128	4,6	205	1738	69	1326	3	63	316	0

TABLE II. CORRELATION TABLE OF E-COMMERCE DOMAIN ATTRIBUTES

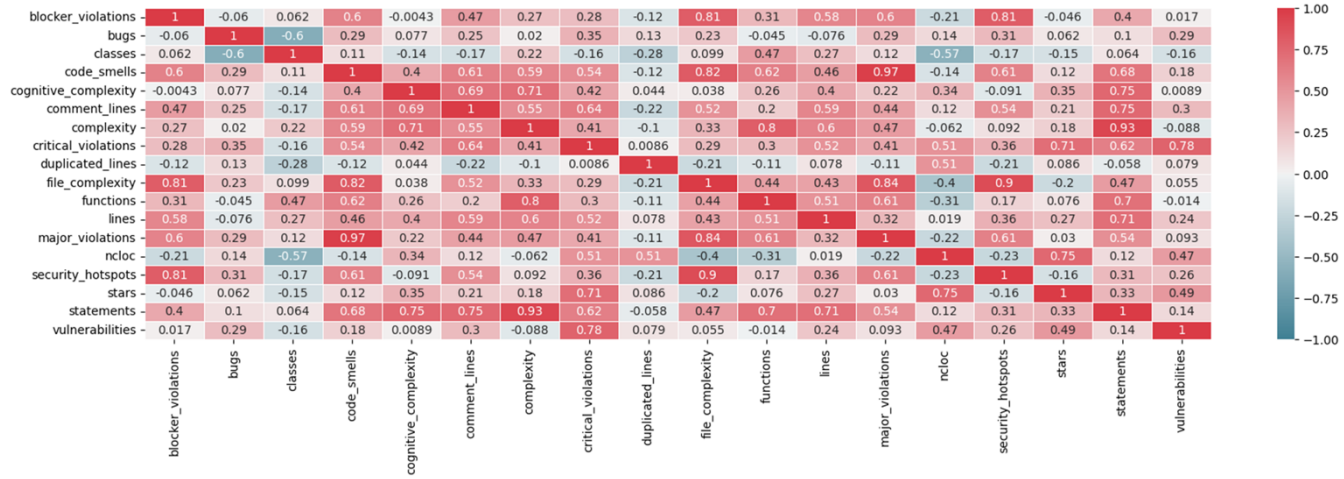

TABLE III. MINED DATA FROM GITHUB FOR GAME DOMAIN

	blocker_violations	bugs	classes	code_smells	cognitive_complexity	comment_lines	complexity	critical_violations	duplicated_lines	file_complexity	functions	lines	major_violations	ncloc	security_hotspots	stars	statements	vulnerabilities
AsciiTerminal	4	16	11	61	404	62	352	10	340	50,3	93	2247	8	1803	3	24	928	0
BatBat-Game	10	21	45	116	633	276	810	7	198	18,8	298	5271	61	3906	12	15	1999	0
gameserver	38	39	475	2126	2387	2749	4739	211	6956	11,7	3225	39201	1204	26089	71	17	10047	3
GameShardingDb	22	21	69	403	730	771	748	38	50	11,5	379	6366	250	4286	7	43	1974	0
jcards	0	0	10	49	44	286	106	0	0	11,8	66	1250	4	509	1	31	163	0
lwjglbook-leg	53	197	1748	3446	9166	5755	20724	349	130553	14,4	13388	155460	1956	116949	203	564	59028	0
OpenFighting	0	35	22	60	81	74	199	33	110	9	147	1559	11	1049	2	21	384	0
playn	43	64	398	1649	2478	6800	6168	163	2431	24,1	4412	46763	807	28753	10	239	11889	0
SypherEngine	3	3	67	233	346	622	731	27	164	9	462	5984	102	3814	4	43	1514	0
WraithEngine	0	1	101	19	307	2283	711	0	0	8,2	577	9081	19	4110	0	50	1289	0

TABLE IV. CORRELATION TABLE OF GAME DOMAIN ATTRIBUTES