Influence of Developer Factors on Code Quality: A Data Study

María José Salamea, Carles Farré
ESSI Department
Universitat Politècnica de Catalunya
Barcelona, Spain
{salamea, farre}@essi.upc.edu

***Abstract*—Automatic source-code inspection tools help to assess, monitor and improve code quality. Since these tools only examine the software project's codebase, they overlook other possible factors that may impact code quality and the assessment of the technical debt (TD). Our initial hypothesis is that human factors associated with the software developers, like coding expertise, communication skills, and experience in the project have some measurable impact on the code quality. In this exploratory study, we test this hypothesis on two large open source repositories, using TD as a code quality metric and the data that may be inferred from the version control systems. The preliminary results of our statistical analysis suggest that the level of participation of the developers and their experience in the project have a positive correlation with the amount of TD that they introduce. On the contrary, communication skills have barely any impact on TD.**

***Index Terms*—code quality, technical debt, human factors, data mining.**

I. Introduction

Digital transformation (DX) —induced by technologies such as agile development, AI (Artificial Intelligence), blockchain, and open APIs, among others— fosters enormous social and economic benefits. DX in software industry demands to reduce time to market and to improve customer experience, operational excellence and quality. These requirements have driven the emergence of software technology applications [1]. For instance, automatic source-code inspection tools are aimed at reducing the human effort that such activity usually requires. On the other hand, Version Control Systems(VCS) help to build, maintain, and evolve projects' codebase. Moreover, VCS store system logs and development data such as commit dates, commit authors, to mention a few.

According to Capgemini [2], human resources are a critical factor even more relevant than technology in DX environments. Particularly, developers have an essential role in software quality [3]. In this regard, studies have shown that developer factors like experience, involvement, compromise, and communication have a direct influence on code quality [4] [5].

In this exploratory study, we assess how developer characteristics impact on code quality. In concrete, we consider the following developer factors: developers' participation, their working experience in the project, and their communications skills. We use Technical Debt (TD) as the metric for code quality. In code terms, TD is introduced by developers whenever they write "bad code". TD impacts negatively on code quality and increases maintainability [6]. Automatic source-code inspection tools typically measure TD, and it represents an appropriate metric because it is calculated as an aggregated of other relevant quality factors [7].

This paper is a preliminary step in a research project whose aim is to improve the level of accuracy of the current state-of-the-art methods and tools in the estimation of the remediation costs - i.e. those costs associated to with the remediation of the code defects. Essentially, it is a first approach to gain knowledge and experience in the tools, models and data that we used in this study.

The rest of this paper is organised as follows. In Section II, we review the literature that analyses the impact of developer factors on code quality. Section III provides the necessary background for the concepts and tools used. In Section IV, we describe the design of our experiments and their results. We address the threats to the validity of our study in Section V. Finally, Section VI presents the conclusions and future work.

II. Related Work

In this section, we present studies that address the relationship between developers characteristics and code quality.

Rahman and Devanbu [4] studied four open source projects to analyse how factors such as developer file ownership, developer file experience, and overall developer experience impact on code quality. They report the following conclusions. Multiple developers contributing to the same file reduce defect code. Developers introduce fewer code defects in their files than in someone else's files. Developer general experience has a weak relationship with code quality.

Alfayez et al. [5] assessed in 38 Apache projects how developer factors such as the developer's frequency of commits, her seniority, and the size of the time lapses between her commits are related to TD. The research results show that the introduction of TD by developers is distributed unequally, with some developers adding more TD than others. This study also determines that there exist a negative correlation between seniority and the TD introduced. Finally, the results show that the higher the time lapse between commits of the same developer is, the higher the TD that she introduces.

Li et al. [8] explored the data of 76 developers in four open software projects in Github to study how the software developers' bug-introducing trends change. Initial results identify two phases in bug introduction: one of increasing and another of decreasing. Moreover, the authors show that the bigger the size of the developer commit is, the higher is the number of bugs introduced.

Qui et al. [9] investigated in six open source projects how developer quality is related to software evolution. The developer quality is measured as the rate of non-bug-introducing commits. The metrics used in this study are the developer contribution calculated as the number of lines of codes that she modifies, and the developer ownership measured as the number of commits that she authors. They concluded that developer quality increases proportionally with software evolution.

III. Background

In this section, we present important concepts considered in our the study.

A. Technical Debt

Cunningham [10] created the Technical Debt (TD) term as an analogy between financial concepts and software cost. There he stated that deficient programming produces debt and it could generate interest in the future. For example, the interest can be extra development time, due to the code is not readable or maintainable. In this way, refactoring represents a mean for paying off this interest. However, refactoring results in a costly process [11]. TD is the total cost to fix issues in the source code, but, at the same time, it can be seen as a mean to address this cost and improve the quality. Although the TD concept is usually applied to the source code, Sierra et al. [12] detected up to five types of TD: design TD, test TD, defect TD, documentation TD, and requirement TD. Therefore, TD can be induced in other parts of the project life-cycle and not just in the implementation and maintenance phases. Still, source code provides relevant information about the quality in previous phases — e.g. flawed requirements, deficient architecture design, etc. Consequently, when the source code is improved, the overall quality increases [13]. Li et al. [14] identify five approaches to TD: prioritization, monitoring, repayment, communication, and prevention. Repayment is the most common approach.

Some methods have been proposed to address TD. Among them, SQALE (Software Quality Assessment based on Lifecycle Expectations) it is worth to mention because it provides a comprehensive TD estimation framework when compared with other methods. For instance, in SQALE, the TD concept encompasses various phases of the development process [15], not only the coding phase. Roughly, the SQALE method inspects the source code to calculate the distance between the current and the target quality. SQALE uses four concepts. The *Quality Model* defines the quality requirements related to the relevant quality characteristics (reliability, efficiency, maintainability, etc.). The *Analysis Model* contains a set of rules to assess quality-requirement violations and calculates TD as the sum of the costs required to reach the "conformity" code. The *Indexes* represents costs. For instance, the extra cost in resources induced by the maintenance activities required to fix a nonconformity. Finally, the *Indicators* provide a visual representation of the TD — e.g., a pyramid indicator depicting the TD distribution per quality characteristics.

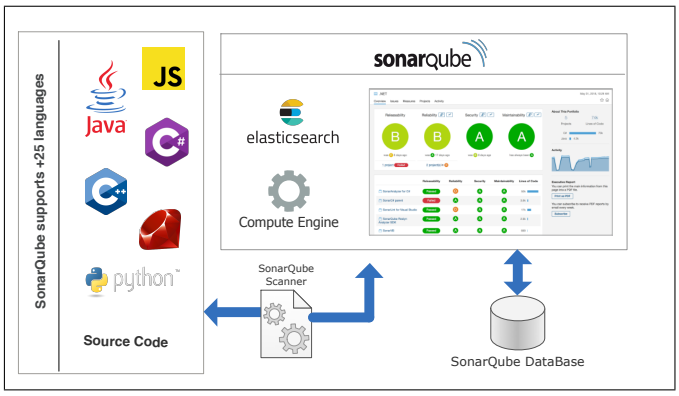

Fig. 1. SonarQube Architecture

B. SonarQube

SonarQube [16] is a automatic inspection code quality tool. To understand how SonarQube works, we describe briefly their components, depicted in Fig. 1. First, the SonarQube-Scanner component enables to launch the analysis of the project code. Second, the SonarQube-Server has a Compute Engine server which performs analysis and reports the status of the project code. It implements the SQALE method to analyse the source code against a set of code rules bounded to specific programming languages called the Quality Profile, to detect code issues and return a software quality diagnostic. This diagnostic reports TD measurements that estimate the effort required to fix the detected maintenance issues. In our study, we use these issues, called TD items, to perform our analysis. The SonarQube-Server uses a Web application to visualise analysis reports and an Elasticsearch server to execute UI requests. Third, the SonarQube-Plugin component allows installing plugins to link SonarQube to another support development tools. Finally, to SonarQube-Database component allows storing the analysis results and configurations.

IV. Experiment

In this Section, we define, first, our Research Questions. Then, we introduce the Github repositories that we use as data sources. We then formulate the data metrics that measure the developer factors we want to correlate with TD. Finally, we describe the data analyses we perform on the data sources and report the obtained results.

A. Research Questions

RQ1: *Is the participation level of the developers in the project related to the amount of TD introduced?* We expect that the developers that write more code are the ones that introduce more TD. However, it can also be expected that

the developers that are more familiar with the project produce proportionality less TD [5]. That is why we formulate next RQ.

RQ2: *Do the developers that have been involved in the project for longer produce less TD than the novice ones?* According to [17] [18], developers with higher participation in a project acquire more expertise, and it could impact on code quality in terms of less TD introduced in code source. For this question, we consider developer experience in the project rather than the developer background.

RQ3: *Do the developers communication skills impact on the efficiency in TD remediation?* Communication factors impact on software quality because efficient communication among developers increases the opportunity to settle correctness issues [19]. Developers use different communication channels such as face-to-face, emails, instant messages, social media, and project management tools, where they comment and discuss decisions, task status, required changes and other subjects. This information contains valuable data: code changes, new issues, bug fixes, and other maintenance tasks. Developers tend to skip comments on changes, but good documentation increases team communication and coordination. That is why we study whether the communication skills of the developers are related in some way to reducing TD.

B. Data Collection

We extract data from the GitHub repositories listed in Table I. For this preliminary study, we have selected two projects with +50 contributors, +5000 commits, Java as the coding language, recent activity, and overall git repository available. From the GitHub repositories, we obtain the following data from each commit as well as the date of the first commit and date of the last commit of each developer:

- Author - name and email
- Deleted and added lines
- Size of the commit message

In SonarQube, each issue represents a unit of work that increases TD, and it is linked to the specific Author (i.e. the developer) who produced it. SonarQube categorises issues as follows:

- Bug - may generate a fault.
- Vulnerability - potential security problem.
- Code Smell - increases maintainability (duplicates, comments, etc.).

In or study, we select from the Sonar Database the number of code smells, i.e. of TD items related to maintainability, that are produced by the different Authors. Finally, we join, by Author, such information with the commit data from Github. In Fig. 2. we show the data extraction process.

C. Metrics Definitions

Table II summarizes the metrics used to analyze the developer factors.

First, to analyze the ***Development Participation*** factor, we decide to use the total lines of code (LOC) edited by a developer. Although this metric is used frequently in literature

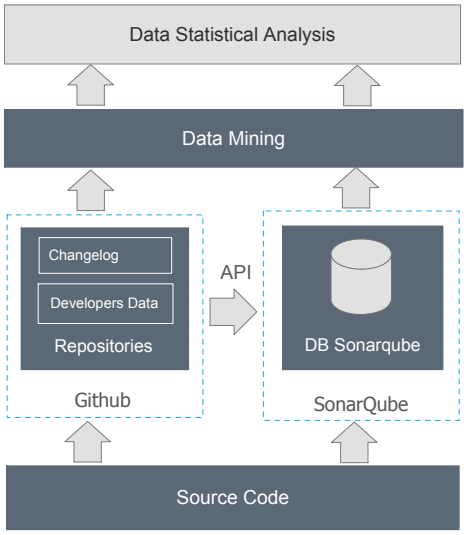

Fig. 2. Data Collection

TABLE I
OPEN SOURCE PROJECTS

Project	Commits	Number of contributors
JabRef	12,527	210
Sonar-Java	6,136	50

[20] [17], there is not a single consensual definition. In our study, we follow the example of [4] and calculate the Total lines of code edited by a developer *i* ($LOC_{edited_{(i)}}$) as follows:

$$LOC_{edited_{(i)}} = LOC_{added_{(i)}} + LOC_{deleted_{(i)}} \quad (1)$$

We measure ***Development experience*** as the number of months ($Time_{part_{(i)}}$) in which a developer *i* participates in the software project. Here, we consider that this number of months spans from the date of first developer commit ($firstcommit_{(i)}$) till the date of last developer commit ($lastcommit_{(i)}$) in the particular project.

We measure ***Developer skill communication*** as the level of detail (size of comments $SoC_{(i)}$) measured in number of lines in a commit.

Finally, ***Code Quality*** is measured in TD items produced by a developer in a software project $TD_{introduced_{(i)}}$. Also, we get number of TD items unresolved by developer $TD_{unresolved_{(i)}}$.

D. Data Analysis

To address our research questions presented in Section IV-A, we analyze the correlation between developer attributes and TD by of means statistical tools commonly applied in the literature. Concretely, we use the Spearman's correlation test [21] to assess the relation between ranked variables. Results are summarized in Table III. In the first column of Table III, we observe the Spearman's correlation coefficient (ϱ) that indicates strength and direction of the association/relationship between evaluated variables. In the second column of Table III,

TABLE II
EVALUATION METRICS

Factor	Metric
Development participation	LOC Edited
Development experience	Time participating in the project
Development skill communication	Size of commit coment

the level of statistical significance (*p-value*) of the correlation coefficient is presented. In this way, if $p < 0.05$ then it means the obtained correlation coefficient value has a statistical significance. To run the Spearman's correlation we use the software SPSS Statistics. Here, the null hypothesis (H_0) states that there is no association between ranked variables ($H_0 : \varrho = 0$). The alternative hypothesis (H_A) states there is an association between ranked variables ($H_A : \varrho \neq 0$). Hypotheses for each RQ are as follows:

To answer RQ1. $H_0 : \varrho = 0$, the correlation coefficient between number of $LOC_{edited_{(i)}}$ and $TD_{introduced_{(i)}}$ is equal to zero. $H_A : \varrho \neq 0$, the correlation coefficient between number of $LOC_{edited_{(i)}}$ and $TD_{introduced_{(i)}}$ is not equal to zero.

For RQ1, we evaluate the relationship between LOC, computed using Eq. 1, and $TD_{introduced}$. In the Fig. 3 we can observe that exist a tendency between LOC_edited and the TD. Here, the Spearman's correlation test shows a positive strong correlation between LOC_edited and $TD_{introduced}$ by each developer (see first column of Table III. Further, we have obtained a significantly lower than 0.05 (see last column of Table III). Therefore, we can accept our H_A (correlated variables). We can conclude there is a statistically significant, strong positive correlation between LOC and TD, $\varrho = 0.7411,\ p = 0.000034$. An increase in LOC was strongly associated with an increase in the TD.

To answer RQ2. $H_0 : \varrho = 0$, the correlation coefficient between $Time_{part_{(i)}}$ and $TD_{introduced_{(i)}}$ is equal to zero. $H_A : \varrho \neq$, correlation coefficient between number of the $Time_{part_{(i)}}$ and $TD_{introduced_{(i)}}$ is not equal to zero.

Here, we evaluate the relation between participation time $Time_{part_{(i)}}$ in the project and TD items introduced per developer. In this case, the Spearman's correlation test shows a positive strong correlation between $Time_{part_{(i)}}$ in the project and TD items introduced per developer. Also, in Table III, we see the Spearman's correlation is statistically significant (i.e., p < 0.05). Therefore, we can accept the H_A (correlated variables). We can conclude there was significant positive correlation between $Time_{part_{(i)}}$ participating in the project and TD items introduced, $\varrho = 0.5439,\ p = 0.006004$.

To answer RQ3. $H_0 : \varrho = 0$, the correlation coefficient between $SoC_{(i)}$ and $TD_{unresolved_{(i)}}$ is equal to zero. $H_A : \varrho \neq$, the correlation coefficient between number of $SoC_{(i)}$ and $TD_{unresolved_{(i)}}$ is not equal to zero.

Finally, in order to test this hypothesis, we evaluate the relation between the size of comments (SoC) and the TD introduced by each developer. In this case, the Spearman's correlation test shows a weaker association between SoC and TD. We can also validate the Spearman's correlation coefficient since the significance (Sig.) is lower than 0.05. We accept H_A and we conclude there was a statistically significant, weak positive correlation between SoC and TD, $\varrho = 0.0072,\ p = 0,000068$.

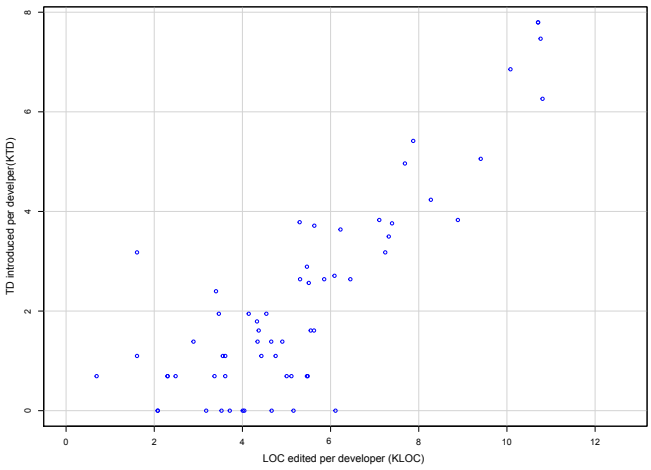

Fig. 3. Correlation between total LOC edited and number of TD produced

TABLE III
VARIABLE CORRELATION RESULTS BY USING SPEARMAN'S CORRELATION TEST

Variable	Correlation Coefficient (ϱ)	*Sig.*
LOC_edited	0.7411	0.000034
$Time_{part_{(i)}}$	0.5439	0.006004
$SoC_{(i)}$	0.0072	0,000068

V. THREATS TO VALIDITY

This Section addresses the most important validity threats that we identify in our study.

A. Conclusion Validity

We use Spearman's correlation test to analyze the relationship between developer factors and code quality. We used it because it is less sensitive to the outliers that exist due to the inherent heterogeneity of the data that comes from different open source projects, as it is reported in another studies [5].

B. Construct Validity

Here our main concern was the accuracy of the tools that we used to assess TD. For mitigating this threat, we selected SonarQube because it is widely accepted by practitioners [22].

C. Internal Validity

We obtained data from open source projects, so our first concern was related to the completeness of data. Thus, we collected data from the start time of the projects to avoid biases and missing data.

Most important, another threat to internal validity comes from the suitability of the metrics that we use to test our hypotheses. As this is an exploratory study, we prioritized the definition of simple metrics and the use of those already present in the literature.

D. External Validity

We acknowledge that this is the most significant threat to the validity of our study. Our conclusions cannot be generalized to other scopes because we considered only two specific Java open source projects. Again, as this is an exploratory study, we prioritized the mastery of the different methods, tools, and data sources.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we present an analysis to determine whether code quality measured in terms of TD is affected by developers factors. We use data from two open source projects, analyzing a total of 18663 commits. We investigate the relation between developer participation, developer experience, and communication skills with TD by using statistical tests.

In the case of developer participation, we find a strong positive correlation between the edited LOC and the amount of TD introduced by each developer, implying that the TD produced is proportional to the LOC edited.

In the second case, we measure the developer experience as time working in a project and the statistical analysis shows that this factor is significantly correlated with TD. However, in order to gain further insight, in future work, we plan to consider also the frequency of commits and the experience of the developers in other similar projects.

Finally, the communication skills of a developer, measured as the number of lines used by a developer to comment on her commits, and the number of TD issues have a weaker association.

As further work, we plan to consider more properly the distinct features of open source projects. In this type of projects, there exists a wide diversity in the developers regarding their involvement degree and collaboration type. Therefore, we plan to characterize different role types to obtain more significant insights. In this regard, we think that could be interesting to track the expertise evolution of those developers with a higher level of participation.

Furthermore, we plan to extend and refine the metrics that we use in this study. For instance, regarding developer experience, we are pondering to take into account the developer's background in other related projects. Regarding, developers' communication skills, we plan to go beyond the measurement of the size of their comments to consider also content analysis.

Finally, we also envision the possibility of integrating the analytical methods that we have introduced in this paper into the framework and tools developed in the Q-Rapids' project [23] [24]. Here, the ultimate aim would be to help to provide accurate estimates on the effort required to remediate code defects induced by developers.

REFERENCES

[1] "Capgemini — World Quality Report 2018-19." [Online]. Available: https://www.capgemini.com/service/world-quality-report-2018-19/

[2] C. Ebert and C. H. Duarte, "Digital transformation," *IEEE Software*, vol. 35, pp. 16–21, 07 2018.

[3] J. Eyolfson, L. Tan, and P. Lam, "Correlations between bugginess and time-based commit characteristics," *Empirical Softw. Engg.*, vol. 19, no. 4, pp. 1009–1039, Aug. 2014. [Online]. Available: http://dx.doi.org.recursos.biblioteca.upc.edu/10.1007/s10664-013-9245-0

[4] F. Rahman and P. Devanbu, "Ownership, experience and defects: A fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 491–500. [Online]. Available: http://doi.acm.org.recursos.biblioteca.upc.edu/10.1145/1985793.1985860

[5] R. Alfayez, P. Behnamghader, K. Srisopha, and B. Boehm, "An exploratory study on the influence of developers in technical debt," in *Proceedings of the 2018 International Conference on Technical Debt*, ser. TechDebt '18. New York, NY, USA: ACM, 2018, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/3194164.3194165

[6] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498 – 1516, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121213000022

[7] L. Lavazza, S. Morasca, and D. Tosi, "Technical debt as an external software attribute," in *Proceedings of the 2018 International Conference on Technical Debt - TechDebt '18*. New York, New York, USA: ACM Press, 2018, pp. 21–30. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3194164.3194168

[8] Y. Li, D. Li, F. Huang, S. Lee, and J. Ai, "An exploratory analysis on software developers' bug-introducing tendency over time," in *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*, Nov 2016, pp. 12–17.

[9] Y. Qiu, W. Zhang, W. Zou, J. Liu, and Q. Liu, "An empirical study of developer quality," in *2015 IEEE International Conference on Software Quality, Reliability and Security - Companion*, Aug 2015, pp. 202–209.

[10] W. Cunningham, Ward, Cunningham, and Ward, "The WyCash portfolio management system," in *Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum) - OOPSLA '92*, vol. 4, no. 2. New York, New York, USA: ACM Press, 1992, pp. 29–30. [Online]. Available: http://portal.acm.org/citation.cfm?doid=157709.157715

[11] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*. New York, New York, USA: ACM Press, 2012, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2393596.2393655

[12] G. Sierra, E. Shihab, and Y. Kamei, "A survey of self-admitted technical debt," *Journal of Systems and Software*, vol. 152, pp. 70–82, 6 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300457

[13] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *Proceeding of the 2nd working on Managing technical debt - MTD '11*. New York, New York, USA: ACM Press, 2011, p. 1. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1985362.1985364

[14] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 3 2015. [Online]. Available: https://www-sciencedirect-com.recursos.biblioteca.upc.edu/science/article/pii/S0164121214002854

[15] J.-L. Letouzey, "The SQALE Method for Managing Technical Debt Definition Document," Tech. Rep., 2016. [Online]. Available: http://www.sqale.org/wp-content/uploads/2016/08/SQALE-Method-EN-V1-1.pdf

[16] "Documentation — SonarQube Docs." [Online]. Available: https://docs.sonarqube.org/latest/

[17] E. Kocaguneli, A. T. Misirli, B. Caglayan, and A. Bener, "Experiences on developer participation and effort estimation," in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, Aug 2011, pp. 419–422.

[18] F. P. Brooks, Jr., *The Mythical Man-month (Anniversary Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[19] A. J. Suali, S. S. M. Fauzi, W. A. W. M. Sobri, and M. H. N. M. Nasir, "Developers' coordination issues and its impact on software quality: A systematic review," in *2017 3rd International Conference on Science in Information Technology (ICSITech)*, Oct 2017, pp. 659–663.

[20] P. Bhattacharya, I. Neamtiu, and M. Faloutsos, "Determining developers' expertise and role: A graph hierarchy-based approach," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sep. 2014, pp. 11–20.

[21] D. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. Boca Raton: Chapman & Hall/CRC, 2011.

[22] L. Pellegrini, "On the Fault Proneness of SonarQube Technical Debt Violations. An empirical study," Ph.D. dissertation, 2018.

[23] L. López, S. Martínez-Fernández, C. Gómez, M. Choraś, R. Kozik, L. Guzmán, A. M. Vollmer, X. Franch, and A. Jedlitschka, "Q-rapids tool prototype: Supporting decision-makers in managing quality in rapid software development," in *Information Systems in the Big Data Era*, J. Mendling and H. Mouratidis, Eds. Cham: Springer International Publishing, 2018, pp. 200–208.

[24] L. Guzmán, M. Oriol, P. Rodríguez, X. Franch, A. Jedlitschka, and M. Oivo, "How can quality awareness support rapid software development? – a research preview," in *Requirements Engineering: Foundation for Software Quality*, P. Grünbacher and A. Perini, Eds. Cham: Springer International Publishing, 2017, pp. 167–173.