2023 IEEE 64th International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS) | 979-8-3503-7029-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ITMS59786.2023.10317724

Machine Learning Powered Code Smell Detection as a Business Improvement Tool

Markuss Siksna
Faculty of Computer Science and Information Technology
Riga Technical university
Riga, Latvia
markuss.siksna@edu.rtu.lv

Ilze Berzina
Faculty of Computer Science and Information Technology
Riga Technical university
Riga, Latvia
ilze.berzina_4@edu.rtu.lv

Andrejs Romanovs
Faculty of Computer Science and Information Technology
Riga Technical university
Riga, Latvia
andrejs.romanovs@rtu.lv

***Abstract* — Code smell represents the level of human interpretability in a software project, which becomes increasingly challenging as modern-day software projects grow in complexity. Machine learning has promising signs of solving the problem of code smell detection but will ultimately be limited by the training dataset of the model. This paper investigates some machine learning approaches for code smell detection, the implications of using such a system, integration with business processes and how such a system would fit into IT governance using Latvia as an example.**

Keywords— business improvement, code smells, machine learning, technical debt

I. Introduction

In today's world modern businesses create and maintain relatively complex software which means that there can be a lot of different interpretations of problems and how they should be handled. Companies expect to benefit and add value throughout the whole product's life cycle. Unfortunately, in IT there can be hidden problems with the product that could show up unexpectedly because of bad practices used during development. These bad practices are called "code smells" which, if unnoticed or ignored turn into technical debt and may cause financial losses to the company.

Code smells or anti-patterns are not bugs, but rather they are a way of measuring the human interpretability of the written code. For instance, large classes, long methods, long parameter lists, high coupling and complex classes could be considered as smelly code [1]. Since code smell is a broad term in some cases it can be subjective, and depending on the project and design pattern, code smells could wary in interpretation [2, 3], but for the purposes of this paper the previously mentioned code smells will be considered as bad practice and overall bad for interpretability.

Code smells can have a relatively huge impact on development and maintenance of code, in some cases anti patterns in code can be the cause of 30% of maintainability problems [4]. Long development times, difficulty debugging and maintaining code are a direct result of code smell and in turn this can lead to indirect impacts on the business side like delayed project deployment and increased costs on maintenance [5, 6, 7].

Since code smells can have a severe indirect impact on business solutions, and since code smell detection can be a time-consuming process [8], it would be in the businesses best interest to swiftly detect and mitigate code smell in their software.

In recent years machine learning has shown the potential of solving a wide range of large, complex, and abstract problems, everything from medical diagnoses to email spam detection, can be solved at least partially with artificial intelligence. That is why artificial intelligence is the prime candidate for tackling this type of problem of code smell detection.

II. ML Approaches for Code Smell Detection

From the reviews examined, code smell detection is mostly approached as a supervised machine learning problem and a wide range of artificial intelligence models have been used to detect code smell [9].

In terms of traditional machine learning algorithms, decision trees (DT) and support vector machines (SVM) have been used the most with varying degrees of success. SVM's work by maximizing the distance between two classes in a hyperplane and DT's work in a tree like structure where a new data entry gets examined for a condition in every node of the tree [11]. For SVM's it has been found that it is relatively hard to regulate hyperparameters and DTs are easier to interpret [10]. The most promising traditional models are usually Random Forests (RF) and JRip with a range of 83-95% F-measure score [9, 10].

Several researchers also have experimented with deep learning models [12, 13, 14]. Models like neural networks (NN) and long short-term memory (LSTM) networks have been experimented with, but the best model usually ends up being convolutional neural networks (CNN) with a F-measure in the range of 89-99% depending on the chosen datasets and selected features [12, 13].

In terms of simplicity traditional machine learning models provide an easier understanding of the model's outcome for the trade-off of some accuracy, but deep learning models work more in a black box manner but provide a more precise result.

As a rule of thumb in modelling, any chosen model for code smell detection ultimately will be limited by its learning data quality and its selected features, this is the main reason why determining the best model is difficult. The best model will most likely differ on a case-by-case basis.

III. Overview of Code Smells and Business Improvement

Another closely related term to code smells is technical debt, also known as code debt. With the arrival of Agile development methods, this phenomenon has caught eye more actively [15]. Technical debt is not always bad or in most cases cannot even be eliminated from the ways of working. Although it is necessary to take actions to minimize it, as it helps improve the business practices [16].

As the biggest limitation for the development project is time, it often comes with consequences like quality issues therefore costs of starting over, unusable software before its expected end of life [17]. This happens when so called technical debt is "not paid off" [15, 18]. This way it directly affects the business financial aspect, needing inclusion of various debt management mechanisms such as cost-benefit analysis [17].

Code smells usually progress and tend persist over time, meaning that businesses goal is to act as soon as possible before they become too costly to afford fixing. It is important to look at both historical and present code when analysing smells. First, it can prevent smells in early versions and second, it will rule out possibility of smells that hide deep behind complex code [16].

Code readability is another quality businesses and individual developers should have good practice of. Bad readability could mean that only the person who wrote the code can understand it, therefore preventing growth and resulting in loss for one of the involved parties [19, 20].

For businesses ignoring code errors means ignoring loss in revenue which could be considered as a potential risk, and if risks are not adequately anticipated and managed could increase the projects likelihood of failure by 27% [21, 22]. A software quality report by CISQ reveals that in 2020 alone US companies combined lost about 2.08 trillion dollars [23]. In 2022 these costs reached 2.41 trillion dollars between operational failures and technical debts [24]. Such operational problems include flight disruptions, major service outages, banking problems and more due to code and overall software related issues. So, it is safe to say that fine-tuning IT businesses is a must.

IV. Implications of using AI for code smell detection

The potential benefits of using machine learning based approaches for code smell detection are somewhat straight forward, lowered development and maintenance periods leads to businesses using less resources which in turn at least theoretically lowers the needed capital for projects.

If ranking is used for machine learning based code smell detection developers could swiftly discover the most acute code smells and prioritize resources to mitigate the smell. And if multi classification is introduced, then developers could find out the type of smell present in the code, which is very valuable information for lowering refactoring times [25].

Since machine learning uses datasets to train and evaluate the models, the datasets used might have some flaws, for example certain biases might be present which in turn can lead to models classifying some code as smelly even though the code is arguably not smelly [1, 10]. Also, if a model uses open uncurated datasets where anyone could contribute, bad actors might undercut the usefulness of the dataset, for example by injecting poorly labelled data in the dataset deliberately or not.

Some datasets used for code smell detection: "PROMISE" which consists of a wide range of projects with labelled data, "CodeXGlue" a dataset which contains a plethora of programming languages in different training and test sizes and "Qualitas Corpus" a curated dataset that contains a vast amount of Java open-source systems [26, 27, 28].

Considering the fact that some projects, for instance, web applications might use multiple programming languages, the complexity of detecting code smells increases, because the machine learning algorithm needs to account for the fact that some design patterns that work in one programming language might not integrate well with a design pattern in a different programming language [4].

V. LLM code generation tools

In recent years Large Language Models (LLM) have facilitated the making of tools that generate code. Using user provided text prompts, these models generate an output sequence. The sequence is generated by a neural network that is usually trained on billions of parameters and multiple hidden layers depending on the model type [29].

Because these models are general purpose models, they can be used to synthesize code. A closed-source tool like GitHub Copilot is specialized for code generation and completion, and other open-source LLM models like GPT-Neo and GPT-J can be specialized to generate code [30, 31].

Although impressive in their feat, these models are still prone to code smells. Models GPT-Neo and GitHub Copilot tend to generate undefined variables, lines that are too long and duplicate code. As for security code smells, the models may use improper check handling for exceptions and produce weak hash functions. But the most prevalent code smell in both model output sequences are undefined variables [31]. Even though code generation LLM's save time for developers, it is advisable to evaluate the code before its deployed into production. At least preliminary tests should be considered and executed beforehand.

VI. Integration with business processes

Although, studies on exact numbers of businesses using code smell detection are limited, it is believed that the practice of use is not widely adopted. As noted in studies, one of the popular reasons for avoidance is variety of programming language use [32]. Another study suggests that it is difficult to validate the results and that false negatives and false positives are diminishing the benefits [1, 33].

Theoretically this could be the reason for the use of machine learning in the process. One of the values regarding the drawbacks is its flexibility. As different machine learning models progress, it increases the possible complexity and range of possible languages used [3].

A survey where developers were asked about code smells points out that quite a large proportion of respondents (32% of 85) did not show any knowledge about code smells [34]. The same survey also revealed that respondents often rely on searched material and code, meaning that used examples may also already contain poor code choices.

It is believed that new developers are the ones making all the smells in the code but often forgotten that most of the code is written by experienced developers. Limiting time and resources even they are prone to this problem and often forced to leave smells in their code [35]. Not controlling development and not refactoring code enough leads to smells that can be left in since the first line of code. A Case study of smell detection within open source microservice applications even shows that system components with smells are historically different from non-smelly systems and exhibit a different change behaviour [36].

Code quality should not be judged by the smells alone. Businesses must implement ways of choosing the right environment, architecture, programming language and more, meaning that aspects like code complexity can vary between different use cases [37]. So more complex code would be better suited for large-scale systems but simpler for smaller applications. This way code maintainability is affected and so are the costs and actions that come with it. As a result, affecting aspects like development lifecycle, project management, quality assurance, time-to-market.

VII. Code smells in IT governance

The CFO of the digital infrastructure service provider company “Stripe” states that collectively, companies today lose upward of $300 billion a year paying down “technical debt”, as developers pour time into maintaining legacy systems or dealing with the ramifications of bad software [38]. The ongoing adjustments that are brought on by user demands or changes in requirement over time create a chain reaction that greatly changes software from the original design and increases its complexity.

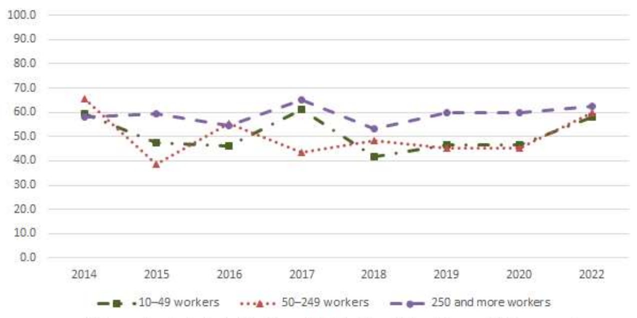
Unfortunately, there are no direct studies relating code smell detection to business improvement or IT governance practices. Instead, there could be found usages of code smell concept in other dimensions of software development and IT governance, for example, architecture, design, implementation, database schema, configuration. To highlight how code smells concept could be used in other dimensions of software development, we conducted a small literature review for enterprise architecture smells detection.

Enterprise architecture (EA) models help organizations establish and maintain effective IT governance by providing a systematic and structured approach to managing IT resources and aligning them with business objectives. Nevertheless, the maintenance of the EA model is often pushed down the priority list due to, e.g., lack of resources or supporting methods. Flaws and deficiencies in the EA may remain ignored and create barriers to EA evolution known as EA debt [39]. Salentin and Hacks connected EA debt with the concept of antipatterns - modelling solution that is known to pose risks - by using code smell taxonomy in EA modelling. Therein, the authors provide a catalogue of 45 EA smells that originated from code smells. In their approach, they transform a catalogue of well-known code smells into EA smells and categorize the EA smells based on the three concerns of EA: business, application, and technology. Furthermore, they present a tool that can detect 14 EA smells. As an extension to their work, this study explores the current knowledge about process anti-pattern to obtain a new understanding thereof in the EA domain. Based on Salentin and Hacks findings in this field Lehmann et.al. also immersed in the study of using EA smell in business process modelling [40, 41] Authors of the study briefly familiarized themselves with another code quality dimension – robustness, which is not one of code smell metrics, but also represents the programmer’s work. Fogen and Lichter concluded a case study related to robustness and raised a question about the usefulness of implementing extra measures of robustness testing and whether they outweigh the additional effort and complexity.[42]

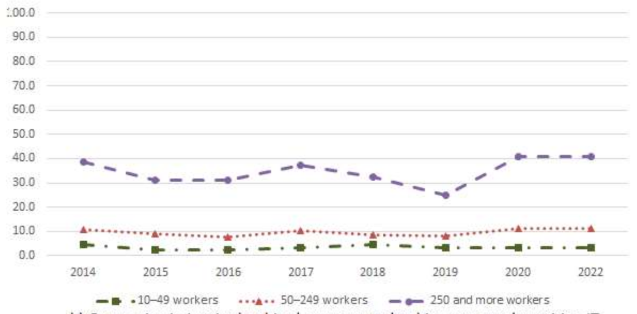
After looking into EA smells, we concluded 2 questions that organizations might have regarding implementing any kind of smells algorithm into their system’s life cycle:

1. Does the code smell or any other kind of smell detection implementation in IT governance outweigh the additional effort and complexity needed to integrate it?
2. As smell detection is designed to minimize technical debt, isn’t there more suitable, already tested and generally suggested solutions for organizations to deal with this kind of debt?

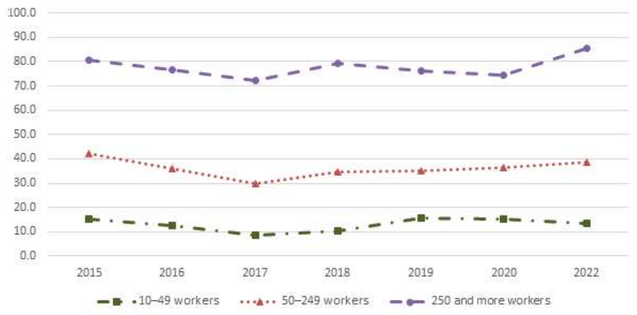
To answer the first question, we decided to study statistics from Information Technology (IT) sector in Latvia. As seen in Fig. 1.c) by year 2022 86.5% of small (10-49 workers), 61.4% of middle sized (50-249 workers) and even 14.4% of big (>250 workers) companies in Latvia does not deem valuable to employ IT specialists. Only 3.1% of small companies and 11.3% of middle-sized companies in 2022 have hired or attempted to hire IT personnel (Fig.1.b). Even more, around 60% of companies of any size faced difficulties trying to hire IT specialists (Fig.1.a). However, Fig.1 d) shows that IT sector in Latvia has kept growing and adding value to GDP, and in 2020 Latvia was one of 7 countries above the EU average - 5.23%. The reasons why companies in Latvia are so shy to employ IT specialists should be a separate study. One of the reasons for such numbers could be that companies choose outsourcing. But even in this case there should be at least someone who is responsible for IT governance in the organization. The other reasons we suggest is that a lot of small and middle-sized companies in Latvia does not see the added value, that digitalization, innovation and IT governance could bring to growth and increased economic competitiveness to the company. Or if it is seen, then it is not chosen to act upon. So, to answer the first question – even if code smell detection would be beneficial in many aspects of IT governance, for most of the companies the additional effort and complexity needed to integrate it is probably deemed to be unworthy.



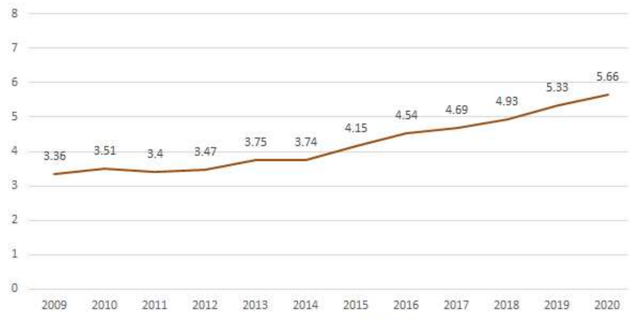
a) Companies in Latvia, that faced difficulties filling IT specialist vacancies (% of companies that hired or attempted to hire personnel)



b) Companies in Latvia that hired or attempted to hire personnel requiring IT specialist skills (% of total companies in the respective group)



c) Companies in Latvia employing IT specialists (% of total companies in the respective group)



d) Percentage of the IT sector on GDP in Latvia (%)

Fig.1. IT sector statistics in Latvia [43, 44]

Answer to the second question is more complex. Technical debt can occur by [45]:

- Pragmatism – when business needs and fast results and income from the product is more important than quality or long-term problems;
- Prioritization – when implementation of critical functions is prioritized over overall quality;
- Processes – lack of communication and collaboration within the team;
- Attitudes - general apathy towards issues of technical debt and software quality, mentality of 'If it ain't broke, don't fix it'.

Tufano M., et.al., on the other hand, studied when and why bad code smells are introduced [46]:

- Most of times code artifacts are affected by bad smells since their creation;
- Code artifacts becoming smelly as consequence of maintenance and evolution activities are characterized by peculiar metrics' trends, different from those of clean artifacts;
- Most code smells are introduced during refactoring operations;
- Newcomers are not necessarily responsible for introducing bad smells, while developers with high workloads and release pressure are more prone to introducing smell instances.

Both technical debt and code smells occur by missing aspects in governance of projects or processes. But it would not be enough to introduce just one framework into IT governance to help this issue. For example, SCRUM methodologies could help with communication, but probably would assist in prioritization of functions over quality and pragmatism. In turn COBIT would help with performance metrics, pragmatism and prioritization, but only partly could help with other code smell or technical debt causes. This is so, because frameworks may dictate how to do, not what to do. So, a possible answer to the second question is – no, there aren't generally accepted IT governance solutions that would solely deal with technical debt and code smells. But what would help is selecting at least some framework to help analyse IT performance and mitigate risks.

VIII. FUTURE DIRECTIONS AND CONCLUSIONS

There are multiple different future directions for code smell detection. One of the biggest opportunities in this field is the creation of code smell detection tools. Special tools that scan the written code detect and highlight potential code smells. Code smell studies and performance evaluations can identify strengths and weaknesses of different detection approaches, that could help developers and businesses choose the most effective tool for their requirements.[1]

Different machine learning application algorithms can be an opportunity to increase code smell detection. Researchers and developers can examine and create new, more advanced, machine learning algorithms and approaches that would be made for code smell detection, that could increase the accuracy and efficiency of code smell identification. But, considering the fact that no such model has yet to be created specifically for code smell detection, a new fresh data-centric approach might be ideal. And the concepts of feature engineering like better labelling and adding new features to the data are integrated to increase the model's performance.[47]

Machine learning code smell detection provides benefits, such as decreased development and maintenance times which in turn should theoretically lower the needed cost of the project. But the machine learning model's effectiveness will be ultimately limited by the provided learning data quality.

There are other future directions and opportunities for code smell detection, like using specialized LLM's, better defining some code smells, integration with multiple languages, better feature selection and dataset protection.

Overall code smells or anti-patterns are well known but sometimes subjective, this fact in turn is one of the main reasons why using machine learning for code smell detection is problematic. In the future there could be a framework that covers the best practices and typical violations regarding code smells or technical debt. Especially considering the possibility of using Large Language Models in code generation, which has still not fully known impact on code smells and technical debt.

Code smell impact on software quality is important, and adopting a code smell detection tool might be a good opportunity for businesses to gain advantages in the long run. Unfortunately, as evident in Latvia's example, there might be more serious underlying problems with IT governance, which might hold companies off implementing new tools [48]. Especially, if the tool's purpose is to prevent something from happening, rather than creating immediate value and its impact is hard to measure. On the other hand, developing and popularizing code smell detection tools could help educate companies about technical debt and possible damage it can cause.

Overall, machine learning powered code smell detection in IT governance could be a powerful tool to improve software development both in time and costs, but for now it is not yet utilized as such. Future studies should be conducted that look at how automatic code smell detection would be fully integrated into businesses software development processes.

REFERENCES

[1] T. Lewowski and L. Madeyski, "Code Smells Detection Using Artificial Intelligence Techniques: A Business-Driven Systematic Review," in Developments in Information & Knowledge Management for Business Applications, N. Kryvinska and A. Poniszewska-Marańda, Eds. Cham: Springer, 2022, vol. 377, Studies in Systems, Decision and Control, pp. 285-319. doi: 10.1007/978-3-030-77916-0_12.

[2] M. V. Mantyla, J. Vanhanen and C. Lassenius, "Bad smells - humans as code critics," 20th IEEE International Conference on Software Maintenance, 2004. Proceedings., Chicago, IL, USA, 2004, pp. 399-408, doi: 10.1109/ICSM.2004.1357825.

[3] M. Lafi, J. W. Botros, H. Kafaween, A. B. Al-Dasoqi and A. Al-Tamimi, "Code Smells Analysis Mechanisms, Detection Issues, and Effect on Software Maintainability," 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 2019, pp. 663-666, doi: 10.1109/JEEIT.2019.8717457.

[4] A. Yamashita and L. Moonen, "To what extent can maintenance problems be predicted by code smell detection? – An empirical study," Information and Software Technology, vol. 55, no. 12, pp. 2223-2242, 2013. doi: 10.1016/j.infsof.2013.08.002.

[5] N. Bessghaier, A. Ouni, and M. W. Mkaouer, "On the Diffusion and Impact of Code Smells in Web Applications," in Services Computing – SCC 2020, Q. Wang, Y. Xia, S. Seshadri, and L. J. Zhang, Eds. Cham: Springer, 2020, vol. 12409, Lecture Notes in Computer Science, pp. 67-84. doi: 10.1007/978-3-030-59592-0_5.

[6] M. Aniche, G. Bavota, C. Treude, et al., "Code smells for Model-View-Controller architectures," Empir. Software Eng., vol. 23, pp. 2121–2157, 2018. doi: 10.1007/s10664-017-9540-2.

[7] F. Khomh, M. Di Penta and Y. -G. Gueheneuc, "An Exploratory Study of the Impact of Code Smells on Software Change-proneness," 2009 16th Working Conference on Reverse Engineering, Lille, France, 2009, pp. 75-84, doi: 10.1109/WCRE.2009.28.

[8] R. Vashisht, "An Empirical Analysis of Code Smell and Code Restructuring in Python," Virtual Technologies and E-Collaboartion for the Future of Global Businesses, Ghaziabad, India, 2023, pp. 203 – 213, doi: 10.4018/978-1-6684-5027-7.ch011

[9] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik and A. De Lucia, "Detecting code smells using machine learning techniques: Are we there yet?," 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, Italy, 2018, pp. 612-621, doi: 10.1109/SANER.2018.8330266.

[10] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," Information and Software Technology, vol. 108, pp. 115–138, 2019. doi: 10.1016/j.infsof.2018.12.009.

[11] B. Haidabrus, E. Druzhinin and O. Psarov, "Taxonomy of Risks in Software Development Projects," 2022 63rd International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS), Riga, Latvia, 2022, pp. 1-7, doi: 10.1109/ITMS56974.2022.9937092.

[12] G. P. Bhandari and R. Gupta, "Measuring the Fault Predictability of Software using Deep Learning Techniques with Software Metrics," 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON), Gorakhpur, India, 2018, pp. 1-6, doi: 10.1109/UPCON.2018.8597154.

[13] S. Tarwani and A. Chug, "Application of Deep Learning models for Code Smell Prediction," 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2022, pp. 1-5, doi: 10.1109/ICRITO56286.2022.9965048.

[14] Y. Zhang, C. Ge, S. Hong, R. Tian, C. Dong, and J. Liu, "DeleSmell: Code smell detection based on deep learning and latent semantic analysis," Knowledge-Based Systems, vol. 255, 2022, Article 109737. Doi: 10.1016/j.knosys.2022.109737.

[15] Z. Codabux and B. Williams, "Managing technical debt: An industrial case study," 2013 4th International Workshop on Managing Technical Debt (MTD), San Francisco, CA, USA, 2013, pp. 8-15, doi: 10.1109/MTD.2013.6608672.

[16] M. Tufano et al., "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," in IEEE Transactions on Software Engineering, vol. 43, no. 11, pp. 1063-1088, 1 Nov. 2017, doi: 10.1109/TSE.2017.2653105

[17] C. Seaman et al., "Using technical debt data in decision making: Potential decision approaches," 2012 Third International Workshop on Managing Technical Debt (MTD), Zurich, Switzerland, 2012, pp. 45-48, doi: 10.1109/MTD.2012.6225999.

[18] Y. Guo et al., "Tracking technical debt — An exploratory case study," 2011 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VA, USA, 2011, pp. 528-531, doi: 10.1109/ICSM.2011.6080824.

[19] M. Osman, "The Business Consequences of Lousy Code" built in. https://builtin.com/software-engineering-perspectives/business-consequences-lousy-code (accessed May. 20, 2023)

[20] R. P. L. Buse and W. R. Weimer, "Learning a Metric for Code Readability," in IEEE Transactions on Software Engineering, vol. 36, no. 4, pp. 546-558, July-Aug. 2010, doi: 10.1109/TSE.2009.70.

[21] P. Goncarovs, "Active Learning SVM Classification Algorithm for Complaints Management Process Automatization," 2019 60th International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS), Riga, Latvia, 2019, pp. 1-3, doi: 10.1109/ITMS47855.2019.8940658.

[22] O. Podzins, A. Romānovs, "IT Risk Identification and Assessment Methodology," Environment. Technology. Resources: Proceedings of the 11th International Scientific and Practical Conference, Rezekne, Latvija, 2017, pp. 124-127, doi:10.17770/etr2017vol2.2539

[23] CISQ, "The cost of poor software quality in the US: A 2020 report," Consortium for Information & Software Quality, Needham, MA, USA, Rep. 1, 2020.

[24] CISQ, "The cost of poor software quality in the US: A 2022 report," Consortium for Information & Software Quality, Needham, MA, USA, Rep. 1, 2022

[25] F. A. Fontana, M. Zanoni, A. Marino and M. V. Mäntylä, "Code Smell Detection: Towards a Machine Learning-Based Approach," 2013 IEEE International Conference on Software Maintenance, Eindhoven, Netherlands, 2013, pp. 396-399, doi: 10.1109/ICSM.2013.56.

[26] S. Lu, et al., "CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation," 2021. [Online]. Available: https://arxiv.org/abs/2102.04664

[27] E. Tempero et al., "The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies," 2010 Asia Pacific Software Engineering Conference, Sydney, NSW, Australia, 2010, pp. 336-345, doi: 10.1109/APSEC.2010.46.

[28] A. M. Lear, E. G. Dada, D. Oyewola, S. Joseph, et. al., "Ensemble Machine Learning Model for Software Defect Prediction," in 2nd International Conference on Software Engineering and Intelligent Systems, 2021, pp. 11-21.

[29] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, A. Oprea, and C. Raffel, "Extracting Training Data from Large Language Models," 2021. [Online], Available: https://arxiv.org/abs/2012.07805.

[30] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A Systematic Evaluation of Large Language Models of Code," arXiv:2202.13169 [cs.PL], Feb. 2022. [Online]. Available: https://arxiv.org/abs/2202.13169. [Accessed: 29 May 2023].

[31] M. L. Siddiq, S. H. Majumder, M. R. Mim, S. Jajodia and J. C. S. Santos, "An Empirical Study of Code Smells in Transformer-based Code Generation Techniques," 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM), Limassol, Cyprus, 2022, pp. 71-82, doi: 10.1109/SCAM55253.2022.00014.

[32] J. Pereira dos Reis, F. Brito e Abreu, G. de Figueiredo Carneiro, et al. “Code Smells Detection and Visualization: A Systematic Literature Review,” Arch Computat Methods Eng, vol 29, pp. 47-94, 2022. [Online]. Available: https://doi.org/10.1007/s11831-021-09566-x

[33] L. Madeyski and T. Lewowski, “Detecting code smells using industry-relevant data,” Information and Software Technology, vol. 155, pp. 107-112, 2023. ISSN 0950-5849. [Online]. Available: https://doi.org/10.1016/j.infsof.2022.107112

[34] A. Yamashita and L. Moonen, "Do developers care about code smells? An exploratory survey," 2013 20th Working Conference on Reverse Engineering (WCRE), Koblenz, Germany, 2013, pp. 242-251, doi: 10.1109/WCRE.2013.6671299.

[35] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto and A. De Lucia, "Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, 2014, pp. 101-110, doi: 10.1109/ICSME.2014.32.

[36] A. Walker, D. Das, and T. Cerny, “Automated Code-Smell Detection in Microservices Through Static Analysis: A Case Study,” Applied Sciences, vol. 10, no 21, pp. 7800, 2020. [Online]. Available: https://doi.org/10.3390/app10217800

[37] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?," 2012 28th IEEE International Conference on Software Maintenance (ICSM), Trento, Italy, 2012, pp. 306-315, doi: 10.1109/ICSM.2012.6405287.

[38] W. Gaybrick. “Tech's ultimate success: Software developers are now more valuable to companies than money”, Sep. 2018. [Online]. Available: https://www.cnbc.com/2018/09/06/companies-worry-more-about-access-to-software-developers-than-capital.html (accessed May. 19, 2023)

[39] S. Hacks, H. Höfert, J. Salentin, Y. C. Yeong and H. Lichter, "Towards the Definition of Enterprise Architecture Debts," 2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW), Paris, France, 2019, pp. 9-16, doi: 10.1109/EDOCW.2019.00016.

[40] B. D. Lehmann, P. Alexander, H. Lichter, and S. Hacks, “Towards the Identification of Process Anti-Patterns in Enterprise Architecture Models,” CEUR Workshop Proceedings, 2757(QuASoQ), 2020, 47-54.

[41] J. Salentin and S. Hacks, “Towards a Catalog of Enterprise Architecture Smells,” 2020 WI2020 Community Tracks. [Online]. Available: https://doi.org/10.30844/wi_2020_y1-salentin

[42] K. Fögen, and H. Lichter “An industrial case study on fault detection effectiveness of combinatorial robustness testing,” CEUR Workshop Proceedings, 2020, 2767(QuASoQ), 29–36.

[43] Use of ICT in households (EPD010). Central Statistical Bureau of Latvia. Oct. 2022. [Online]. Available: https://stat.gov.lv/lv/statistikas-temas/informacijas-tehn/e-prasmes/tabulas/epd010-ikt-it-specialisti-uznemumos

[44] Percentage of the ICT sector on GDP [TIN00074__custom_6367106]. Eurostat the statistical office of the European Union. May. 2023. [Online]. Available: https://ec.europa.eu/eurostat/databrowser/view/TIN00074__custom_6367106/default/line?lang=en

[45] 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. ISSN 0164-1212. [Online]. Available: https://doi.org/10.1016/j.jss.2012.12.052

[46] M. Tufano et al., "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," in IEEE Transactions on Software Engineering, vol. 43, no. 11, pp. 1063-1088, 1 Nov. 2017, doi: 10.1109/TSE.2017.2653105.

[47] T. Sharma, A. Jatain., S. Bhaskar, and K. Pabreja, "Ensemble Machine Learning Model for Software Defect Prediction," Procedia Computer Science, vol. 218, pp. 199 - 209, 2023. ISSN 1877-0509 [Online]. Available: https://doi.org/10.1016/j.procs.2023.01.002

[48] A. Teilans, A. Romanovs, J. Merkurjevs, P. Dorogovs, A. Kleins, S. Potryasaev, "Assessment of Cyber Physical System Risks with Domain Specific Modelling and Simulation," SPIIRAS Proceedings, vol. 4, no. 1, pp. 115-139, 2018, doi:10.15622/sp.59.5