DACOS—A Manually Annotated Dataset of Code Smells

Himesh Nandani, Mootez Saad, Tushar Sharma
Dalhousie University
Halifax, Canada
{himesh.nandani, mootez, tushar}@dal.ca

***Abstract*—Researchers apply machine-learning techniques for code smell detection to counter the subjectivity of many code smells. Such approaches need a large, manually annotated dataset for training and benchmarking. Existing literature offers a few datasets; however, they are small in size and, more importantly, do not focus on the subjective code snippets. In this paper, we present DACOS, a manually annotated dataset containing $10,267$ annotations for $5,192$ code snippets. The dataset targets three kinds of code smells at different granularity–*multifaceted abstraction, complex method,* and *long parameter list*. The dataset is created in two phases. The first phase helps us identify the code snippets that are potentially subjective by determining the thresholds of metrics used to detect a smell. The second phase collects annotations for potentially subjective snippets. We also offer an extended dataset DACOSX that includes definitely benign and definitely smelly snippets by using the thresholds identified in the first phase. We have developed TAGMAN, a web application to help annotators view and mark the snippets one-by-one and record the provided annotations. We make the datasets and the web application accessible publicly. This dataset will help researchers working on smell detection techniques to build relevant and context-aware machine-learning models.**

I. INTRODUCTION

Code smells are symptoms of poor design and implementation [1]. Existing literature shows that code smells have a negative impact on maintainability [2], [3], development effort [4], [5], and reliability [6]–[9] among other quality attributes. Given its importance, the software engineering community has put significant effort to study various dimensions, such as their characteristics, impact, causes, and detection mechanisms, related to code smells [10].

Many code smells are subjective in nature [10] *i.e.,* a snippet may exhibit a smell in one context, but a similar snippet may not be considered smelly in another context. *Context* includes the used programming language, experience of the development team, and quality-related practices followed in an organization. A simple example of smells' subjectivity is a method with, for example, 80 lines of code. Based on the context, it could be a *large method* for some developers; others might not classify the method as a large method. However, a method with 500 lines of code will be *definitely* a large method for all developers.

Currently, the majority of commonly used tools use metrics and heuristics to identify code smells [10]. It is often argued that due to the subjective nature of smells, one cannot come up with universally accepted metric thresholds to classify a snippet in a smelly or benign code in all contexts. [10], [11] To overcome the challenge introduced by the subjective nature of smells, researchers propose smell detection using machine-learning techniques [11]–[16]. Such approaches rely on a code smells dataset, ideally manually annotated, to train a machine-learning model. However, existing datasets offer little on multiple fronts. First, the literature offers only a handful of datasets such as LANDFILL [17]. Second, existing datasets contain a small number of annotated samples; for example, LANDFILL offers annotations for only 243 snippets. A dataset with a small number of annotated samples would help a little to train state-of-the-art deep-learning models with reasonable accuracy. Next, existing code smells datasets do not filter out code snippets that are definitely benign or smelly. For example, a snippet with a very few (say, three) lines of code cannot have a *long method*; similarly, a snippet with a very large (*e.g.,* 200) number of lines of code definitely suffers from a *long method* smell. Given that the *value*, in terms of effectiveness, of a smell dataset lies in the captured subjectivity, such definite snippets, either definite benign or smelly, reduce the efficacy offered by a dataset. Lastly, the available support for different types of smells is limited; for example, LANDFILL offers annotated snippets for five types of smells. Given the huge amount of effort involved in annotating code snippets, the software engineering community needs to complement existing smell datasets for other kinds of actively researched smells.

In this paper, we offer a manually annotated dataset of code smells *viz.* ***DAtaset of COde Smells*** (DACOS). To create an effective dataset, we filtered the code snippets that are likely to be subjective by removing the snippets that are either definitely benign or smelly. This approach helps us better utilize the annotators' effort by considering their inputs where we actually need them. The dataset offers annotated code snippets for three code smells— *multifaceted abstraction* [18], [19], *complex method* [20], and *long parameter list* [1]. In addition to a manually annotated dataset on potentially subjective snippets, we offer DACOSX dataset containing a large number of snippets that are either definitely benign or smelly. Furthermore, we developed a web-application *viz.* TAGMAN to make it easy for annotators to see one snippet at a time, and indicate whether a smell is present in the snippet. We have made source code of TAGMAN[1] available publicly.

[1]https://github.com/SMART-Dal/Tagman

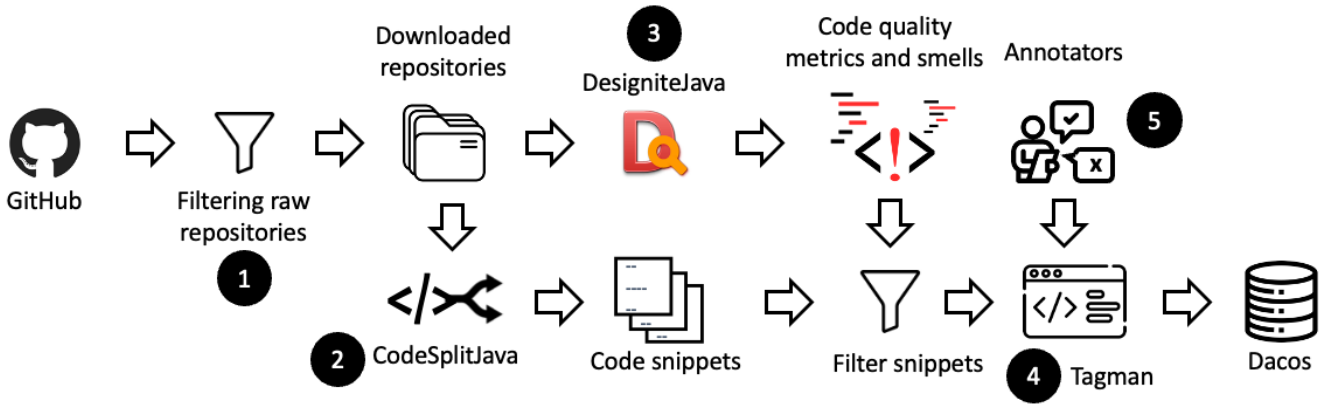

Fig. 1. Dataset construction process

We make the following contributions to the state of the art.

- We offer DACOS, a manually annotated code smell dataset, containing 10,267 annotations for 5,192 code snippets for the considered code smells.
- We also provide DACOSX, an extended dataset containing 207,605 snippets that are either definitely benign or smelly. These datasets will help researchers in the field to train and validate their machine-learning models.
- A configurable web application TAGMAN for easy smell annotations. The community may use the application for similar code annotation purposes.

II. DATASET CONSTRUCTION

Figure 1 provides an overview of the dataset construction process. We elaborate on the steps in detail below.

A. Downloading repositories

In step 1 from Figure 1, we perform the following tasks to identify and download repositories.

- We use `searchgithubrepo` [21] python package, which in turn uses the GITHUB GRAPHQL API [22] to filter GITHUB repositories.
- To identify high quality Java repositories, we select repositories with more than or equal to 13 thousand stars and more than ten thousand lines of code.
- Also, we discard the repositories that are not modified in the last one year.
- In addition, we use QScored [23] to filter out repositories based on their code quality score. QScored assigns a weighted quality score based on the detected smells at various granularities. We select repositories that have a quality score less than ten (the higher the score, the poorer the quality).
- Finally, we obtained ten repositories after applying the filtering criteria. We download the selected repositories.

B. Dividing the repositories into classes and methods

We need to split a repository into individual methods and classes so that TAGMAN can show individual snippets one by one to an annotator. We use CODESPLITJAVA [24] in step 2 to split each repository into individual methods and classes.

C. Analyzing repositories

In step 3, we employ a metrics-based filtering process in the phase-2 of manual annotation. We use DESIGNITEJAVA [25] to compute code quality metrics. DESIGNITEJAVA computes a variety of code quality metrics and detects smells; it has been used in various studies [26]–[30]. We elaborate the process to filter out non-subjective samples in the manual annotation step.

D. Tagman

TAGMAN, shown as 4 in Figure 1, is a web-based tool that we created to facilitate the annotation process. Figure 2 shows a screenshot of the application showing a code snippet and an option to annotate the snippet with a smell. The front-end of TAGMAN is written in *Thymeleaf* and HTML/CSS. The back-end of the application is developed in *SpringBootJava* and the data is stored in a MYSQL database. Figure 3 shows the schema of the database.

At the beginning of the code smell annotation cycle, we upload a CSV file containing the repository names and URL of selected GITHUB repositories. TAGMAN back-end uses a set of Python scripts[2] to download GITHUB repositories, split the code into class and method files, and run DESIGNITEJAVA.

Once the data import is completed, the tool is ready to start accepting annotations. To start an annotation session, a user login (or sign up) to the application. Then, the user is presented with instructions including the definitions of code smells. The user can then start annotating the presented code snippets.

E. Manual annotation

We employ a snippet selection mechanism to identify potentially subjective snippets *w.r.t.* a code smell. We do so to improve the effectiveness of the resultant dataset by only including manual annotations for potentially subjective code

[2]https://github.com/SMART-Dal/Tagman-scripts

Fig. 2. Annotation user interface of TAGMAN

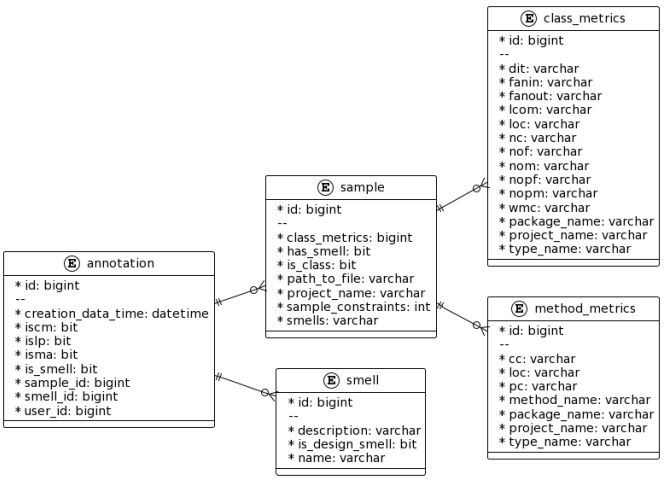

Fig. 3. Schema of the DACOS database

snippets. Also, such a strategy helps us better utilize the available annotators' time. A potentially subjective snippet is a code snippet that may get classified as benign or smelly based on context. The rest of the snippets that are not identified as potentially subjective snippets are either definitely benign or smelly snippet. For example, *cyclomatic complexity* (CC) [31] is commonly used to detect *complex method* smell. A code snippet is definitely benign if CC is very low (*e.g.,* CC=1); similarly, a snippet is definitely smelly if CC is very high for a method (*e.g.,* CC=30). We divide our annotation process into two phases. In the first phase, we show all snippets, *i.e.,* without any filtering, to annotators to identify metrics thresholds to determine whether a snippet is potentially subjective or not. The second phase uses the identified metrics thresholds and show the filtered code snippets to annotators.

1) Phase-1: In the first phase, we show code snippets without any filtering to annotators. TAGMAN presents one snippet at a time to the annotators and collects their response on whether the shown snippet has a code smell or not. We show a code snippet to two randomly chosen annotators and record their responses. The annotators recruited, on a volunteer basis, for this phase were graduate students of Computer Science enrolled in a software engineering course (during summer 2022) that cover code smells extensively. A total of 110 annotators participated in this phase.

We compute the minimum and maximum threshold for metrics that are used to decide the presence of a code smell based on the collected responses in Phase-1. We received a total $17,869$ responses in this phase. We compute the lowest metric value (t_l) where the smell is identified, for each smell individually, to obtain the threshold at the lower side. Similarly, we extract the highest metric value (t_h) where the smell is *not* identified. Then, we compute the standard deviation (`sd`) of the metric value for the samples where the smell is identified. Finally, we obtain the low threshold using $\texttt{max}(m_l, t_l - \texttt{sd})$ and high threshold using $\texttt{min}(m_h, t_h + \texttt{sd})$ for subjective snippet identification. Here, m_l and m_h represent the lowest and highest possible values of a metric. Table I summarizes the quality metrics corresponding to each code smell and their thresholds for identifying subjective snippets. For instance, for *cyclomatic complexity* metric, we obtain 4 and 7 from the above calculation after rounding the values to the nearest integer.

TABLE I
CODE QUALITY METRICS USED FOR CODE SMELLS AND THEIR LOW AND HIGH THRESHOLDS FOR SUBJECTIVE SNIPPET IDENTIFICATION

Code smell	Code quality metric	Metric threshold
Complex method	Cyclomatic complexity	4–7
Long parameter list	Parameter count	3–6
Multifaceted abstraction	Lack of cohesion among methods (LCOM)	0.4–0.8

2) Phase-2: We configure our filtering mechanism based on the thresholds obtained from Phase-1 and invite annotators by advertising the link of TAGMAN installation on social media platforms such as Twitter and LinkedIn. The invitation was open to all software developers, software engineering students, and researchers who understand Java programming language and at least basic object-oriented concepts. We kept the invitation open for six weeks during Dec-Jan 2022-23. A total of 82 annotators participated in this phase. TAGMAN showed snippets that have metric values falling between the low and high thresholds (inclusive). We configured TAGMAN to get two annotations for each sample to improve the reliability of the annotations.

F. Dataset information

After the phase-2, we received a total of $10,267$ annotations for $5,192$ samples from 86 annotators. Table II presents the number of annotations and samples per smell type.

Dataset availability: The datasets offered in this paper are available online [32]. Also, the repositories containing scripts[2] used to prepare code snippets as well as code annotation application[3] *i.e.,* TAGMAN are available online.

[3]https://github.com/SMART-Dal/Tagman

TABLE II
DATASET METADATA

Dataset	Code smell	#Annotations	#Samples
DACOS	Complex method	4,349	2,197
	Long parameter list	3,221	1,634
	Multifaceted abstraction	2,697	1,361
	Total	**10,267**	**5,192**
DACOSX	Complex method	–	94,489
	Long parameter list	–	93,442
	Multifaceted abstraction	–	19,674
	Total	**–**	**207,605**

III. POTENTIAL RESEARCH APPLICATIONS

- **Detecting and validating code smells:** Given the subjective nature, traditional smell detection tools that implement rules and heuristics to identify smells, do poorly. The success of a machine-learning approach to detect smells depends on the availability of a large manually annotated dataset. The presented datasets, DACOS and DACOSX, complement existing datasets by offering a large number of subjective code snippets for three code smells that the existing datasets do not cover.
- **Correlating code smells with software engineering aspects:** A variety of exploratory and empirical studies investigating the impact of code smells exists. It includes bug prediction [33], maintainability prediction [34], and maintenance effort [35]. In addition to existing directions, the tools trained or fine-tuned from the offered datasets can be used to effectively establish a relationship between code smells and productivity of a software development team.
- **Extending TAGMAN for code annotation:** The code annotation application that we developed to create this dataset can be extended for similar kinds of code annotation, for example, to spot vulnerable code and to segregate well-written identifiers.

IV. RELATED DATASETS

Software engineering literature offers a small number of manually annotated datasets for code smells. Palomba *et al.* [36] offered a dataset LANDFILL containing annotations for five types of code smells—*divergent change, shotgun surgery, parallel inheritance, blob,* and *feature envy*. They offered annotations for 243 snippets. They also developed an online portal where contributors can annotate code for smells, however as of writing this paper (Jan 2023), the portal is not accessible. Madeyski *et al.* [37] proposed MLCQ— a code smell manually annotated dataset. The dataset contains 14.7 thousand annotations for $4,770$ samples. The dataset considered four smells—*blob, data class, long method,* and *feature envy*. Both of the datasets mentioned above do not consider the subjectiveness of a code snippet; hence most of the snippets might not add any new information for a machine-learning classifier when used in training. Also, we chose the code smells that are not covered by any existing code smell dataset and hence complement the existing datasets. There are some code smells datasets such as QScored [23]. Though the QScored dataset is large, the samples are not manually annotated and hence lack the required capturing of context.

V. THREATS TO VALIDITY

Internal validity threats concern the ability to draw conclusions from our experimental results. In phase-2 of manual annotation, we invited volunteers with at least a basic understanding of Java programming language and object-orientation concepts. We advertised the invitation on social media professional channels (Twitter and LinkedIn). Given the anonymity of the exercise, we do not have any mechanism to verify the assumption that the participants has sufficient knowledge to attempt the annotations. However, we offered all the major participants (with at least 50 annotations) to include them as contributors to the dataset; we perceive such a measure would have motivated the annotators to perform the annotations to the best of their abilities. Additionally, we configured TAGMAN to obtain two annotations per sample so that we can reduce the likelihood of a random annotation.

External threats are concerned with the ability to generalize our results. The proposed dataset is for snippets written in Java. However, our code annotation tool is generic and it can be used to annotate snippets from any programming language. Additionally, scripts used to generate individual snippets can be customized to use any other external tool for splitting the code into methods and classes. Furthermore, the thresholds used in the annotation process to filter snippets based on low and high thresholds of a metric are configurable.

VI. LIMITATIONS, CONCLUSIONS, AND FUTURE WORK

We offer DACOS—a manually annotated code smell dataset containing $10,267$ annotations for $5,192$ subjective code snippets. We also providea large DACOSX dataset containing definitely benign and definitely smelly snippets in addition to those present in DACOS. The paper offers TAGMAN, a code annotation application, that can be reused in similar contexts.

The proposed dataset covers three code smells. We selected a rather small set of code smells to consider in the dataset because it is better to have more number of annotations for a smell rather than having small number of annotations per smell. Also, we chose a set of smells that are not covered by existing code smells dataset. We configured TAGMAN to obtain two annotations per sample. Though it improves the reliability of the dataset, one may argue that it may introduce a situation where the annotations are contradictory to each other. We can mitigate the limitation by increasing the number of annotations per sample to three; we will incorporate this mechanism in the future version of the datasets. Additionally, we would like to expand the scope of the dataset in terms of programming language, number of samples, and number of supported smells in the future.

REFERENCES

[1] M. Fowler, *Refactoring: Improving the Design of Existing Programs*, 1st ed. Addison-Wesley Professional, 1999.

[2] M. Perepletchikov and C. Ryan, "A controlled experiment for evaluating the impact of coupling on the maintainability of service-oriented software," *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 449–465, Aug. 2011.

[3] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley, "An empirical analysis of the distribution of unit test smells and their impact on software maintenance," in *IEEE International Conference on Software Maintenance, ICSM*, Universita di Salerno, Salerno, Italy. IEEE, Dec. 2012, pp. 56–65.

[4] D. I. Sjøberg, A. Yamashita, B. C. Anda, A. Mockus, and T. Dybå, "Quantifying the effect of code smells on maintenance effort," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1144–1156, 2013.

[5] Z. Soh, A. Yamashita, F. Khomh, and Y.-G. Guéhéneuc, "Do Code Smells Impact the Effort of Different Maintenance Programming Activities?" in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2016, pp. 393–402.

[6] A. Ikama, V. Du, P. Belias, B. A. Muse, F. Khomh, and M. Hamdaqa, "Revisiting the impact of anti-patterns on fault-proneness: A differentiated replication," in *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2022, pp. 56–65.

[7] F. Khomh, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change- and fault-proneness," *Empirical Software Engineering*, vol. 17, no. 3, pp. 243–275, Jun. 2012.

[8] T. Hall, M. Zhang, D. Bowes, and Y. Sun, "Some Code Smells Have a Significant but Small Effect on Faults," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, pp. 33–39, Sep. 2014.

[9] F. Jaafar, Y.-G. Guéhéneuc, S. Hamel, and F. Khomh, "Mining the relationship between anti-patterns dependencies and fault-proneness," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, Ecole Polytechnique de Montreal, Montreal, Canada. IEEE, 2013, pp. 351–360.

[10] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, vol. 138, pp. 158 – 173, 2018.

[11] U. Azadi, F. A. Fontana, and M. Zanoni, "Poster: Machine learning based code smell detection through wekanose," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018, pp. 288–289.

[12] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia, "Comparing heuristic and machine learning approaches for metric-based code smell detection," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 93–104.

[13] 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?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 612–621.

[14] A. Imran, "Design smell detection and analysis for open source java software," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 644–648.

[15] T. Sharma, V. Efstathiou, P. Louridas, and D. Spinellis, "Code smell detection by deep direct-learning and transfer-learning," *Journal of Systems and Software*, vol. 176, p. 110936, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221000339

[16] T. Sharma, M. Kechagia, S. Georgiou, R. Tiwari, I. Vats, H. Moazen, and F. Sarro, "A survey on machine learning techniques for source code analysis," 2022. [Online]. Available: https://arxiv.org/abs/2110.09610

[17] F. Palomba, D. D. Nucci, M. Tufano, G. Bavota, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "Landfill: An open dataset of code smells with public evaluation," May 2015. [Online]. Available: https://doi.org/10.5281/zenodo.6080422

[18] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, 1st ed. Morgan Kaufmann, 2014.

[19] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 189–200. [Online]. Available: https://doi.org/10.1145/2901739.2901761

[20] ——, "House of cards: Code smells in open-source c# repositories," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 424–429.

[21] T. Sharma, "searchgithubrepo - search github repositories," https://github.com/tushartushar/search-repo, 2022.

[22] "Github graphql api documentation." [Online]. Available: https://docs.github.com/en/graphql

[23] T. Sharma and M. Kessentini, "Qscored: A large dataset of code smells and quality metrics," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 590–594.

[24] T. Sharma, "Codesplitjava," Feb. 2019, https://github.com/tushartushar/CodeSplitJava. [Online]. Available: https://doi.org/10.5281/zenodo.2566865

[25] T. Sharma, P. Mishra, and R. Tiwari, "Designite - a software design quality assessment tool," in *2016 IEEE/ACM 1st International Workshop on Bringing Architectural Design Thinking Into Developers' Daily Activities (BRIDGE)*, 2016, pp. 1–4.

[26] W. Oizumi, L. Sousa, A. Oliveira, L. Carvalho, A. Garcia, T. Colanzi, and R. Oliveira, "On the density and diversity of degradation symptoms in refactored classes: A multi-case study," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 346–357.

[27] T. Sharma, P. Singh, and D. Spinellis, "An empirical investigation on the relationship between design and architecture smells," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4020–4068, 2020.

[28] A. Eposhi, W. Oizumi, A. Garcia, L. Sousa, R. Oliveira, and A. Oliveira, "Removal of design problems through refactorings: Are we looking at the right symptoms?" in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 148–153.

[29] A. Uchôa, C. Barbosa, W. Oizumi, P. Blenilio, R. Lima, A. Garcia, and C. Bezerra, "How does modern code review impact software design degradation? an in-depth empirical study," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 511–522.

[30] M. Alenezi and M. Zarour, "An empirical study of bad smells during software evolution using designite tool," *i-Manager's Journal on Software Engineering*, vol. 12, no. 4, p. 12, 2018.

[31] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[32] H. Nandani, M. Saad, and T. Sharma, "Dacos - dataset," Jan. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7570428

[33] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. Oliveto, "Smells like teen spirit: Improving bug prediction performance using the intensity of code smells," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 244–255.

[34] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 306–315.

[35] D. I. K. Sjøberg, A. Yamashita, B. C. D. Anda, A. Mockus, and T. Dybå, "Quantifying the effect of code smells on maintenance effort," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1144–1156, 2013.

[36] F. Palomba, D. Di Nucci, M. Tufano, G. Bavota, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Landfill: An open dataset of code smells with public evaluation," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 482–485.

[37] L. Madeyski and T. Lewowski, "Mlcq: Industry-relevant code smell data set," in *Proceedings of the Evaluation and Assessment in Software Engineering*, ser. EASE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 342–347. [Online]. Available: https://doi.org/10.1145/3383219.3383264