Journal of Data Acquisition and Processing
ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.777794

PYCORPUS: A BENCHMARK CORPUS OF PYTHON PROGRAMS FOR DYNAMIC PROGRAM ANALYSIS

Amit Kumar Dogra [a], *, Harsh Kumar Verma [a]
[a] Dept. of Computer Science and Engineering, Dr B R Ambedkar National Institute of Technology Jalandhar, Punjab, India
*amitkumar62003@gmail.com

ABSTRACT

The popularity of DaCapo and SPEC JVM in both academia and industry can be used to gauge the significance of benchmark suits in the field of software engineering. However, the majority of the benchmark sets that are available in the existing body of knowledge, are either based on Java, C++, or C hence missing out on python which is one of the fastest growing language of the contemporary programming world. This study introduces PyCorpus, a collection of 15 executable Python programs, in an effort to fill the gap left by the lack of a Python-specific benchmark suite in the field of Dynamic Program analysis. In order to investigate, assess, and compare the performance, efficiency, and dynamic elements of the Python language, PyCorpus seeks to offer scholars and practitioners with a thorough collection of Python programs that exhibits dynamic behaviors.

KEYWORDS: Benchmark, DaCapo, SPEC, Dynamic Analysis, Python, Program analysis, Dynamic program analysis.

1. INTRODUCTION

Benchmark suites play a crucial role in software analysis, providing standardized and representative programs that enable researchers and practitioners to evaluate and compare the performance of different software systems. The availability of benchmark suites allows for fair and objective performance assessments, facilitates the identification of bottlenecks, and aids in the development of optimization techniques. While benchmark suites have been widely developed for various programming languages, including Java and C, there has been a notable absence of a comprehensive benchmark suite for Python programs. Moreover most of the existing benchmark available in the literature have been developed for static analysis of softwares[1][2][3][4]. Static analysis is often assumed to be inherently sound but has limited precision in maintaining soundness. However, certain aspects of a program are not captured by static analysis, making it unsound in real-world scenarios. Dynamic programming language features, such as reflection, proxies, class loading, and binding, are difficult to capture by static analysis techniques. Many studies have pointed out the inefficacy of static analysis of softwares especially when it comes to design quality assessment of softwares as it is hugely impacted by dynamic features of programming languages like inheritance, late binding etc. [5][6][7][8].

1.1. Motivation

Python has gained immense popularity in recent years, becoming one of the most widely used programming languages in various domains[9], including web development, data analysis[10], Artificial Intelligence[11] etc. However, the absence of a standardized benchmark suite for Python limits the ability to conduct accurate and reliable performance evaluations, compare different Python programs, and identify areas for optimization[12]. To address this gap, we introduce PyCorpus, a comprehensive benchmark suite tailored specifically for dynamic design analysis of Python programs.

1.2. Importance of Benchmark Suites in Software Analysis

Benchmark suites serve as valuable resources for researchers and practitioners involved in software analysis[12][13]. They provide a standardized set of programs that enable fair performance evaluations, aid in identifying performance bottlenecks[14], fault predictions[15], code comprehension[16] and facilitate the development and validation of optimization techniques. Benchmark suites contribute to the advancement of software engineering by enabling the comparison of different systems, the identification of best practices, and the evaluation of new technologies[17].

1.3. Research Goals and Objectives

The primary goal of this research is to introduce PyCorpus, a comprehensive benchmark suite for Python programs, and address the absence of a standardized benchmark suite in the Python ecosystem. The specific objectives of this research article are as follows:

- Construct PyCorpus using a rigorous and systematic methodology
- Provide a diverse set of representative Python programs covering various domains and application types
- Facilitate accurate performance evaluations and comparisons of Python programs
- Enable the identification of performance bottlenecks and optimization opportunities
- Foster community engagement and contributions to enhance PyCorpus over time

2. RELATED WORK

2.1. Existing Benchmark Suites and their Significance

DaCapo: The DaCapo Benchmark Suite[18] is a collection of open-source Java benchmarks designed to evaluate the performance of Java Virtual Machines (JVMs) and compilers. It consists of a set of real-world applications that cover a range of different computational workloads. The suite includes various benchmarks that simulate different types of applications, such as compilers, database systems, XML processing, object serialization, and more. The DaCapo Benchmark Suite has been widely used by researchers and developers to compare and evaluate the performance of JVMs, as well as to analyze the effectiveness of optimization techniques and compiler optimizations.

SPECjvm: SPECjvm[19] is a benchmark suite designed by SPEC (Standard Performance Evaluation Corporation) specifically for evaluating the performance of Java Virtual Machines (JVMs) and Java applications. The SPECjvm benchmark suite consists of a set of representative Java applications and benchmarks that cover various aspects of Java performance, including computational and memory-intensive workloads. It aims to provide a standardized methodology for evaluating JVM performance across different hardware and software configurations.

The Qualitas Corpus[20] is a large-scale software benchmark and dataset that has been widely used for empirical studies and evaluations in software engineering research. It was created to support research on software quality, maintainability, and related aspects. The corpus consists of a collection of open-source Java projects, providing a diverse set of software systems from various application domains.

XCorpus[21], a set of 76 executable, real-world Java programs, includes a subset of 70 programs from the Qualitas Corpus. The data set uses a harness that combines built-in and generated test cases, resulting in a branch coverage significantly better than what is available from DaCapo.

Name	Year	Language	Programs	Version History	Harness
DaCapo	2006	Java	14	No	Yes
SpecJVM	2008	Java	10	Yes	No
Qualitas Corpus	2008	Java	112	Yes	No
XCorpus	2017	Java	70	Yes	Yes
PyCorpus	2023	Python	15	Yes	No

2.2. Benchmark Suites for Java and Other Programming Languages

Benchmark suites have been widely developed for various programming languages to facilitate performance evaluations and comparisons. One prominent example is the DaCapo benchmark suite for Java, which has had a significant impact on Java software analysis. DaCapo comprises a diverse set of programs that cover a wide range of application domains, allowing researchers to assess the performance of Java virtual machines and identify potential optimizations.

2.3. Importance of Existing Benchmark Suites

Existing benchmark suites, such as DaCapo for Java, have played a crucial role in advancing software analysis by providing standardized benchmarks. These suites have significantly contributed to the evaluation of runtime performance, memory utilization, scalability, and responsiveness of software systems. They have enabled the identification of performance bottlenecks, the comparison of different optimization techniques, and the validation of new approaches in software engineering research.

2.4. Gap in Benchmark Suites for Python

While benchmark suites have been developed for several programming languages, including Java, C, and others, there has been a noticeable absence of a standardized benchmark suite specifically tailored for Python programs. This gap limits the ability to accurately evaluate and compare the performance of Python software systems, hindering advancements in the field. The absence of a comprehensive benchmark suite for Python calls for the development of PyCorpus to address this critical need.

2.5. Significance of PyCorpus

PyCorpus aims to fill the gap in the absence of a standardized benchmark suite for Python programs. By providing a comprehensive and diverse set of benchmarks, PyCorpus enables researchers and practitioners to conduct accurate performance evaluations, compare different Python programs, and identify areas for optimization. The significance of PyCorpus lies in its ability to foster fair and objective performance assessments, facilitate the development of optimization techniques, and promote best practices in Python software engineering.

2.6. Building on Existing Benchmarks

While existing benchmark suites for other programming languages[22][14] provide valuable insights and methodologies, PyCorpus takes into account the unique characteristics and dynamic features of Python. It incorporates benchmarks that capture the specific behaviors and performance considerations of Python programs, allowing for a more accurate assessment of Python software systems. PyCorpus builds upon the lessons learned from existing benchmark suites while tailoring its benchmarks to the specific requirements and nuances of the Python programming language.

The existing benchmark suites, such as DaCapo for Java, have demonstrated the importance of standardized benchmarking in software analysis. However, the absence of a comprehensive benchmark suite for Python programs has limited the ability to conduct accurate performance evaluations and comparisons. PyCorpus aims to bridge this gap by providing a diverse set of benchmarks specifically tailored for Python, enabling researchers and practitioners to advance the field of Python software engineering through fair and objective performance assessments, optimization opportunities, and best practice identification.

3. METHODOLOGY

The construction of PyCorpus, a comprehensive benchmark suite for Python programs, involved a systematic and rigorous methodology to ensure the selection of representative benchmarks and the integration of associated test cases. This section outlines the key steps and considerations undertaken during the construction process.

3.1. Benchmark Selection

The first step in constructing PyCorpus was to identify a wide range of Python programs that would serve as representative benchmarks. To achieve this, we considered multiple sources, including popular open-source projects, research codebases, and real-world

applications. These sources were carefully selected to cover various domains, programming paradigms, and libraries commonly used in Python development.

During the benchmark selection process, we paid particular attention to capturing the dynamic aspects of the Python language. We aimed to include benchmarks that utilize dynamic features such as late binding, polymorphism, dynamic typing, and reflection. By encompassing these dynamic characteristics, PyCorpus ensures that the benchmark suite accurately reflects the behavior and performance considerations unique to Python programs.

To ensure the quality and relevance of the selected benchmarks, we evaluated them based on several criteria. These criteria included the popularity and significance of the projects, the maturity and stability of the codebase, the availability of multiple versions, and the presence of associated test cases. By considering these factors, we aimed to include benchmarks that would provide meaningful insights into the performance characteristics and evolution of Python software systems.

3.2. Test Case Integration

An integral part of PyCorpus is the integration of test cases with each benchmark. Test cases serve as a means to validate the correctness and functionality of the benchmarked programs and provide a standardized set of inputs for performance evaluations. The integration of test cases ensures that PyCorpus not only focuses on performance but also covers the functional aspects of the software systems.

The process of test case integration involved studying the existing test suites associated with the selected benchmarks and identifying relevant test cases. We considered both unit tests and functional tests that exercise different parts of the programs and cover various usage scenarios. The test cases were adapted and integrated into PyCorpus, ensuring their compatibility and consistency with the benchmarked programs.

3.3. Versioning and Evolution

Another important aspect of PyCorpus is the inclusion of multiple versions of each benchmark program. This allows researchers to study the evolution of software over time and analyze the impact of changes, improvements, or regressions on performance characteristics. Collecting multiple versions involved identifying different releases or revisions of the benchmarked projects and capturing the corresponding codebases.

By including multiple versions, PyCorpus enables researchers to assess the performance implications of software evolution and provides insights into the effectiveness of optimization techniques and development practices. It allows for comparative analysis between different versions and facilitates a better understanding of the performance trade-offs associated with software changes.

3.4. Documentation and Dissemination

PyCorpus is accompanied by comprehensive documentation that provides detailed information about the benchmark selection process, program characteristics, associated test cases, performance metrics, and usage guidelines. The documentation serves as a guide for researchers and practitioners interested in utilizing PyCorpus for their performance evaluations and optimization efforts. In addition to documentation, the dissemination of PyCorpus is crucial for its adoption and impact. We have made PyCorpus openly available to the Python community through a public repository[23].

By following this methodology, we have successfully constructed PyCorpus, a comprehensive benchmark suite that captures the dynamic aspects of Python programs. PyCorpus provides researchers and practitioners with a valuable tool for performance analysis, optimization, and software engineering advancements in the Python ecosystem.

4. OVERVIEW OF PYCORPUS

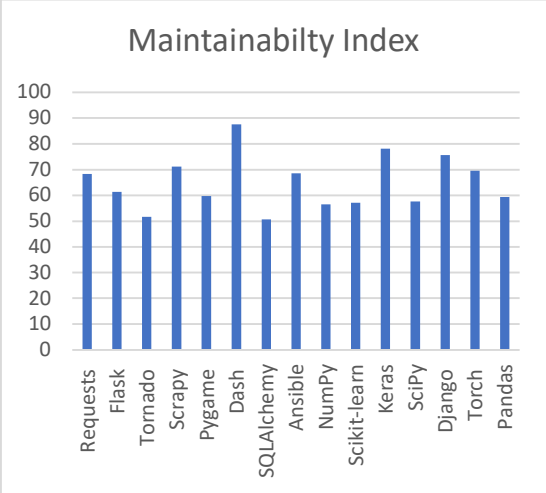
PyCorpus includes a wide range of representative benchmarks carefully selected from various sources. These benchmarks cover different domains, programming paradigms, and libraries commonly used in Python development. By encompassing this diversity, PyCorpus ensures that researchers and practitioners have access to benchmarks that accurately reflect the real-world scenarios and challenges encountered in Python programming.

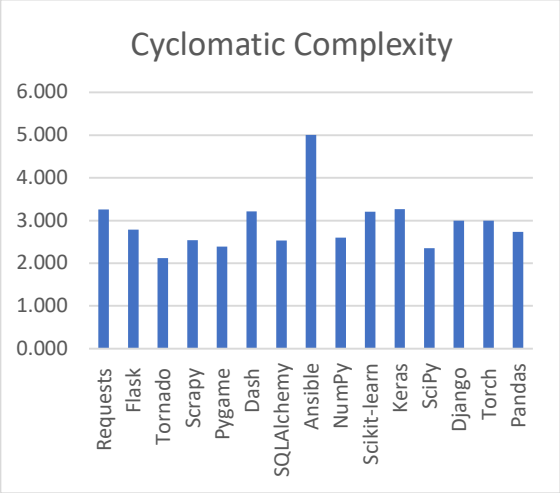
PyCorpus	
1. Requests - HTTP library	9. NumPy - Numerical computing library
2. Flask - Micro web framework	10. Scikit-learn - Machine learning library
3.Tornado - asynchronous networking library	11. Keras - Deep learning library
4. Scrapy - Web scraping framework	12. SciPy - Scientific computing library
5. Pygame - Game development library	13. Django - Web framework
6. Dash - Data visualization framework	14. Torch - Deep learning library
7. SQLAlchemy - Object-relational mapper	15. Pandas - Data analysis libraryPandas
8. Ansible - IT automation tool	

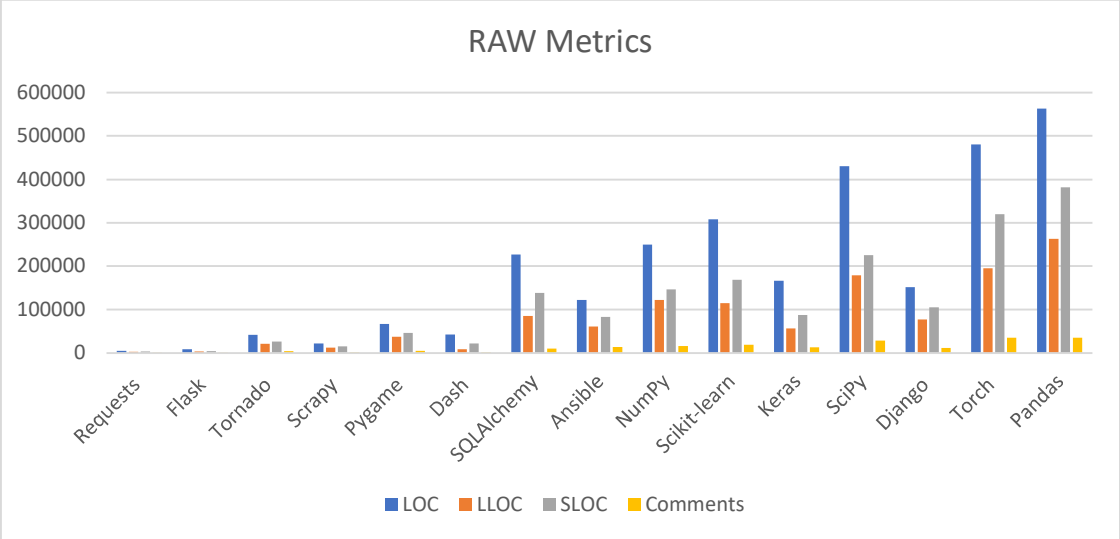
It serves as a valuable resource for researchers and practitioners involved in performance analysis, optimization, and software engineering advancements in the Python ecosystem. PyCorpus encompasses a diverse set of representative benchmarks, associated test cases, and dynamic metrics to facilitate accurate and meaningful evaluations of Python software systems.

Some of the basic features/metrics like Cyclomatic Complexity(CC), Maintainability Index(MI), and Raw metrics (LOC, LLOC, SLOC) etc. of these projects as reported by radon[24] has been given below.

Project	Blocks	CC	Classes	mi	LOC	LLOC	SLOC	Comments
Requests	272	3.261	18	68.375	5435	2458	2891	450
Flask	376	2.787	22	61.394	8791	3247	3920	700
Tornado	3074	2.115	72	51.623	42034	21118	25857	4079
Scrapy	1702	2.536	170	71.144	22219	12331	15207	753
Pygame	3163	2.390	172	59.766	67050	37429	46106	5230
Dash	919	3.214	207	87.613	42913	8412	21721	1037
SQLAlchemy	11546	2.532	252	50.643	226767	84964	138112	9921
Ansible	4523	5.004	479	68.551	122756	60828	82997	13409
NumPy	12080	2.598	484	56.468	249832	122068	146715	15636
Scikit-learn	9338	3.212	548	57.078	308286	114728	168687	19263
Keras	5315	3.266	604	78.093	165770	56389	88005	12692
SciPy	18483	2.350	797	57.624	430421	179165	225487	28672
Django	10329	2.998	872	75.674	151296	77456	105302	11632
Torch	20847	2.997	1362	69.63	480941	195483	319828	34986
Pandas	26965	2.729	1371	59.276	563651	263410	381353	35337







5. DYNAMIC ASPECTS CAPTURED BY PYCORPUS

The programs in PyCorpus are chosen to capture the dynamic aspects of the Python language. They leverage features such as late binding, polymorphism, dynamic typing, and reflection, which are fundamental to Python's flexibility and expressiveness. By including these dynamic features, PyCorpus enables researchers to analyze and understand the performance implications of Python's unique language characteristics.

6. USECASES AND APPLICATION SCENARIOS

PyCorpus, as a curated collection of FOSS Python programs with specific attributes, can serve as a valuable resource for various use cases and application scenarios in empirical studies, benchmarking, machine learning, and software engineering education.

6.1. Empirical Studies:

Comparative Analysis: Researchers can perform comparative studies on different projects within PyCorpus to understand patterns, best practices, and common pitfalls in Python software development.

Code Quality Analysis: PyCorpus can be used to analyze code quality metrics across projects, identify trends, and assess the impact of various coding styles and practices on maintainability and performance.

Dynamic Feature Analysis: Researchers can study the prevalence and usage patterns of dynamic features in Python by examining code samples from PyCorpus.

Evolutionary Analysis: PyCorpus's projects with multiple versions can be analyzed to study code evolution, identify changes in design patterns, and assess the impact of refactoring.

6.2. Benchmarking Tools and Techniques:

Tool Comparison: Researchers can use PyCorpus to benchmark different static analysis tools, testing frameworks, and performance analysis tools to evaluate their effectiveness and efficiency on real-world projects.

Code Coverage Analysis: PyCorpus can be used to benchmark code coverage tools and techniques to assess their accuracy in measuring test coverage.
Dynamic Analysis Techniques: Researchers can compare various dynamic analysis techniques (e.g., profiling, tracing) on PyCorpus to understand their effectiveness in different scenarios.

6.3. Training Machine Learning Models:

Machine Learning for Code Analysis: Researchers can use PyCorpus to train machine learning models for code classification, bug detection, and other software engineering tasks.
Predictive Maintenance: Machine learning models trained on PyCorpus can help predict maintenance needs, identify code smells, and anticipate potential issues in software projects.

6.4. Enhancing Software Engineering Education:

Learning Resources: PyCorpus can be a valuable resource for educational purposes, providing real-world examples for students to study, analyze, and learn from.
Code Review Practice: Students can practice code review on projects from PyCorpus to improve their code analysis and critical thinking skills.
Best Practices: PyCorpus can serve as a reference for demonstrating best practices, design patterns, and coding conventions in Python development.

6.5. Language Feature Exploration:

Language Research: PyCorpus can be used to explore how different Python language features are used in real-world projects, providing insights into the popularity and practicality of specific language constructs.

6.6. Automated Code Generation and Refactoring:

PyCorpus can serve as a dataset for training machine learning models to automatically generate code snippets or assist in code refactoring tasks.
Overall, PyCorpus offers a diverse range of Python projects that can be leveraged for research, analysis, benchmarking, and educational purposes. The availability of multiple versions of projects allows for historical analysis and trend identification. Researchers, practitioners, and educators can benefit from PyCorpus to advance their understanding and expertise in Python software engineering.

7. LIMITATIONS AND FUTURE DIRECTIONS

While PyCorpus represents a significant contribution to the field of Python software analysis, there are several avenues for future work and improvements:

7.1. Expansion of Benchmark Suite

One area for future work is the expansion of PyCorpus to include an even broader range of benchmarks. This expansion can encompass additional domains, programming paradigms, and libraries, ensuring a more comprehensive representation of the Python ecosystem. By including

a wider variety of benchmarks, PyCorpus will become an even more powerful tool for performance analysis and optimization.

7.2. Inclusion of Real-World Applications

Another direction for future work is the inclusion of benchmarks based on real-world Python applications. This would involve collaborating with industry partners or open-source projects to incorporate their software systems into PyCorpus. By including real-world applications, PyCorpus can provide researchers and practitioners with insights into the performance characteristics of production-grade Python programs.

7.3. Integration with Existing Tools and Frameworks

PyCorpus can be further enhanced by integrating it with existing tools and frameworks used for performance analysis and optimization in the Python ecosystem. This integration would allow researchers and practitioners to leverage PyCorpus seamlessly within their existing workflows and take advantage of the features and capabilities offered by these tools.

7.4. Community Collaboration and Contributions

A vital aspect of future work is fostering community collaboration and contributions to PyCorpus. By establishing PyCorpus as an open-source project, researchers and practitioners can contribute new benchmarks, test cases, and performance metrics based on their specific needs and areas of expertise. Community involvement will ensure the continuous improvement and relevance of PyCorpus over time.

The future work outlined above will further enhance the capabilities of PyCorpus and contribute to the ongoing improvement and development of Python software analysis tools and technique

8. CONCLUSION

PyCorpus represents a significant contribution to the field of Python software analysis by providing a comprehensive benchmark suite tailored for Python programs. With its diverse set of representative benchmarks, integration of test cases, inclusion of multiple versions, and dynamic metrics, PyCorpus enables researchers to gain insights into the performance behavior of Python programs and make informed decisions regarding accurate performance evaluations, optimization opportunities, and software engineering advancements. Practitioners can leverage PyCorpus to identify performance bottlenecks, optimize critical sections of their codebase, or assess the performance implications of software changes. The availability of PyCorpus as a standardized benchmark suite fosters collaboration, enables comparative studies, and promotes advancements in the field of Python software analysis.

REFERENCES

[1] M. J. Ordonez and H. M. Haddad, “The State of Metrics in Software Industry,” in *Fifth International Conference on Information Technology: New Generations (itng 2008)*, 2008, pp. 453–458.

[2] S. R. Chidamber and C. F. Kemerer, “A Metrics Suite for Object Oriented Design,” *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

[3] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc., 1994.

[4] F. Brotoeabreu, "The MOOD Metrics Set," in *Proc. ECOOP'95 Workshop Metrics*, 1995.

[5] N. E. Fenton and M. Neil, "Software metrics: successes, failures and new directions," *J. Syst. Softw.*, vol. 47, no. 2–3, pp. 149–157, 1999.

[6] M. D. Ernst, "Static and dynamic analysis: synergy and duality," *WODA 2003 ICSE Work. Dyn. Anal.*, pp. 24–27, 2003.

[7] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 672–681.

[8] K. Goseva-Popstojanova and A. Zahalka, "The impact of dynamic metrics on identification of the failure prone parts of the software," *Lane Dept. Comput. Sci. Electr. Eng. West Virginia Univ. Morgantown, USA, June*, 2006.

[9] S. Oveflow, "Stack Overflow developer survey 2023." [Online]. Available: https://survey.stackoverflow.co/2023/#most-popular-technologies-language-prof.

[10] W. McKinney, *Python for data analysis*. " O'Reilly Media, Inc.," 2022.

[11] R. Zulunov and B. Soliev, "Importance of Python language in development of artificial intelligence," *Потомки Аль-Фаргани*, vol. 1, no. 1, pp. 7–12, 2023.

[12] S. E. Sim, S. Easterbrook, and R. C. Holt, "Using benchmarking to advance research: A challenge to software engineering," in *25th International Conference on Software Engineering, 2003. Proceedings.*, 2003, pp. 74–83.

[13] P. Botman, "Software Assessments, Benchmarks and Best Practices," *Softw. Qual. Prof.*, vol. 4, no. 2, p. 48, 2002.

[14] A. Meneely, B. Smith, and L. Williams, "Validating Software Metrics: A Spectrum of Philosophies," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 4, pp. 1–28, 2012.

[15] S. Beecham, T. Hall, D. Bowes, D. Gray, S. Counsell, and S. Black, "A Systematic Review of Fault Prediction approaches used in Software Engineering," *Engineering*, no. 03, 2010.

[16] H. A. Valdecantos, K. Tarrit, M. Mirakhorli, and J. O. Coplien, "An Empirical Study on Code Comprehension: Data Context Interaction Compared to Classical Object Oriented," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, 2017, pp. 275–285.

[17] W. Hasselbring, "Benchmarking as empirical standard in software engineering research," in *Evaluation and Assessment in Software Engineering*, 2021, pp. 365–372.

[18] S. M. Blackburn *et al.*, "The DaCapo Benchmarks : Java Benchmarking Development and Analysis ∗," in *21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2006, pp. 169–190.

[19] K. Shiv, K. Chow, Y. Wang, and D. Petrochenko, "SPECjvm2008 Performance Characterization," in *Computer Performance Evaluation and Benchmarking*, 2009, pp. 17–35.

[20] E. Tempero *et al.*, "The Qualitas Corpus: A curated collection of Java code for empirical studies," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010, pp. 336–345.

[21] J. Dietrich, H. Schole, L. Sui, and E. Tempero, "XCorpus – An executable Corpus of Java Programs (preprint , under review for jot . fm)," pp. 1–26, 2017.

[22] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, "Source code metrics: A systematic mapping study," *J. Syst. Softw.*, vol. 128, pp. 164–197, 2017.

[23] Amit Kumar Dogra and H. K. Verma, "PyCorpus Github repository." [Online]. Available: https://github.com/amitkumar62003/PyCorpus.git.

[24] "Radon." [Online]. Available: https://radon.readthedocs.io/en/latest/index.html#. [Accessed: 25-Jun-2023].