COMP3023J Software Methodology Research Group11

1st Jinpeng Zhai
Software Engineering Year of 2021
Beijing-Dublin International College
Beijing, China
jinpeng.di@ucdconnect.ie

2nd Ziqi Yang
Software Engineering Year of 2021
Beijing-Dublin International College
Beijing, China
ziqi.yang@ucdconnect.ie

3rd Yiran Zhao
Software Engineering Year of 2021
Beijing-Dublin International College
Beijing, China
ZhaoYiran@emails.bjut.edu.cn

***Abstract*—This paper, authored by Group 11, serves as a midterm checkpoint for the COMP3023J Software Methodology course research assignment. It is important to note that a comprehensive abstract will be provided upon the completion of the entire manuscript.**

***Index Terms*—Python, tracing, visualization**

I. Introduction

Python is widely recognized as one of the most popular programming languages in contemporary computing. Its popularity stems from its exceptional flexibility, ease of learning, and the extensive selection of libraries available [1]. Python's widespread adoption is not limited to well-established fields such as web development, data analytics, and scientific computing; it has also gained prominence in emerging areas, notably in the domain of machine learning [2]. This increasing popularity has led to a growing community of developers and researchers.

However, as Python projects expand in size and complexity, they encounter fresh challenges associated with both development and maintenance. These challenges encompass intricacies in code organization, the emergence of "hot code" leading to performance bottlenecks, and the escalating intricacies of ongoing maintenance [3]. Consequently, there arises a compelling need for code analysis and optimization for large Python projects [4].

Previous studies have emphasised the efficacy of employing code tracing and code coverage techniques within Python projects in tackling these problems. Code tracing refers to the action of following the flow of program execution. It shares some similarities with deterministic profiling and debugging, in that in the context of Python, this is usually accomplished via placing a hook function into the Python interpreter via the provided *sys.settrace()* method, that is then called to record the exact detail of program execution. Among these, CPython's Lib.trace.py module provides robust capabilities in tracing the execution of native Python code [5]. This module adeptly traces function calls and furnishes comprehensive code coverage reports, thus facilitating the identification of inadequately tested code and performance bottlenecks.

Simultaneously, a growing trend in recent years has witnessed the adoption of visualization tools to enhance code analysis, presenting findings in an intuitively accessible format [6], [7]. This trend serves to further enhance code quality and bolster the maintainability of large Python projects.

Of note is that due to the converging interest in developing better techniques for the visualization of profiling results, in recent years popular visualization packages have instead opted for extending either the *pdb*, the Python debugger module, or various deterministic profilers, traditionally either the *profile* or the *cProfile* packages. The merits of some of these options will be explored later in this report.

Consequently, this research aims to explore and compare various options for tracing Python code visualization, and demonstrate the use of these techniques for a better understanding of complex Python projects.

II. Previous Art

Lib/trace.py is the tool bundled with CPython, the reference implementation for Python language [8]. Written in the early 2000s, it has been part of the official library from as early as Python 2.2. It provides both program execution tracing and code coverage analysis functionality, and even a limited profiling ability. Implementation-wise, it attaches a tracing funciton (via *sys.settrace()*), written in Python, that is attached to the evaluation mechanism in the underlying Python interpreter, and is called during the normal execution of CPython [9]. Although the module itself does not provide for much in terms of visualization, in recent years a few projects have emerged that rectified this lapse. This work will examine one such project, *vizTracer*.

Various Python profilers, in particular profiling tools bundled with CPython as part of the standard library, the *profile* and the C implementation *cProfile* are also of some interest. As deterministic profilers, they records and expose the Python call stack for each evaluation, allowing program execution tracing to be inferred. Due to the wealth of information recorded, over the years, many projects that visualizes program execution has built upon its output. Notable ones includes:

1) *RunSnakeRun*. Generates a variation of tree map using performance statistics. Call relation implied by geometrical relations.
2) *GProf2Dot* and *PyCallgraph*. Generates a directed arrow-and-box graph with performance statistics, broken down based on call relations.

Of note is that not all profilers are suitable for our purposes. In particular, non-deterministic profilers works by sampling, at set intervals, the memory layout of the Python interpreter, thus they are suitable for low overhead programm profiling, but unsuitable when capturing function calls for analysis are required, since there is no guarantee of the completeness of the resulting capture. Solutions based on these will be ignored for the purpose of this study.

Finally it bears mentioning that an emerging technique employed in the visualizaiton of tracing information is that of static code analysis. In particular, sophisticated models have been proposed in academica to analyze Python code that has demonstrated limited ability to infer call relation without running the code. However, due to the dynamically typed, interpreted nature of Python, this approach has met some challenges when met with complex code structure, dynamic or conditional imports, and deep class hierarchies.

III. Methodology

Describe the way that you do your own research and how you are going to explore the thesis statement that you made.

IV. Experiments and Results

Analyze experimental results.

V. Discussion

Explore the results and talk about the implications that the results have upon the entire body of research which existed.

VI. Conclusion

General statement and ending thoughts about the topic.

VII. Project Management

Throughout the course of this research project, effective project management practices were diligently implemented to ensure the successful execution of the study and the attainment of its research objectives. This section offers insights into key aspects of project management, including team management, the presentation of milestones through Gantt charts, individual team member contributions, and potential future research directions.

A. Team Management

Effective team management is paramount for the success of a research project, ensuring that all team members work cohesively towards common goals. In the context of this study, several strategies were meticulously employed to promote efficient teamwork:

- **Cohesive and Collaborative Team:** Our research team thrived on a culture of cohesiveness and collaboration. Team members actively engaged in learning about each other's work, encouraging knowledge sharing, and fostering an environment of continuous improvement. Mutual respect, inclusivity, and honesty were the cornerstones of our interactions. We recognized and celebrated each team member's distinct attributes and strengths, resulting in a balanced and complementary team.
- **Regular Team Meetings:** The project team scheduled and adhered to regular meetings to facilitate thorough discussions on various aspects of the research. These meetings became a platform for deliberating project progress, addressing issues, and setting forth objectives for the upcoming stages. The consistency of these meetings ensured that all team members were continuously aligned with the research objectives and were aware of the project's evolving dynamics.
- **Effective Team Organization:** Organizational structures significantly impact the decision-making process within a team. Effective communication is the cornerstone of our team management strategy. In addition to scheduled meetings, we established digital communication platforms, specifically WeChat and Telegram group chats, to facilitate real-time communication and information sharing among team members. These platforms allowed for swift discussions and collaborative decision-making, ensuring that everyone was on the same page regarding project developments.
- **Shared Resource Management:** To facilitate efficient collaboration and resource accessibility, the research team adopted modern tools and platforms. A dedicated GitHub repository was established for streamlined code sharing and collaboration on experimental components. This central repository allowed team members to work on project software with version control and simplified issue tracking. Additionally, Overleaf, an online platform for collaborative writing, was used to create and edit the research paper collectively. This real-time collaboration improved the efficiency of document creation and review. These modern tools ensured easy access to project resources, including code and research documentation, enhancing collaboration, version control, and issue resolution for the success of the research project.

Our team management practices were instrumental in fostering a collaborative and productive environment, which ultimately played a pivotal role in the successful execution of this research project. The harmonious teamwork, communication, and resource management strategies allowed us to efficiently navigate the complexities of the research process and achieve our project objectives.

B. Milestone Showcase with the Gantt Chart

As illustrated in Figure 1, the Gantt chart is an instrumental graphical representation that offers a structured overview of the project's chronological progression and key milestones. This visual representation is an essential tool for effective project management, aiding in the tracking of tasks, allocation of resources, and adherence to project timelines.

C. Team Member Contributions

Each member of the research team made substantial and equitable contributions to the success of the project. It is worth noting that the team unanimously acknowledges that each member's contributions to the research project were

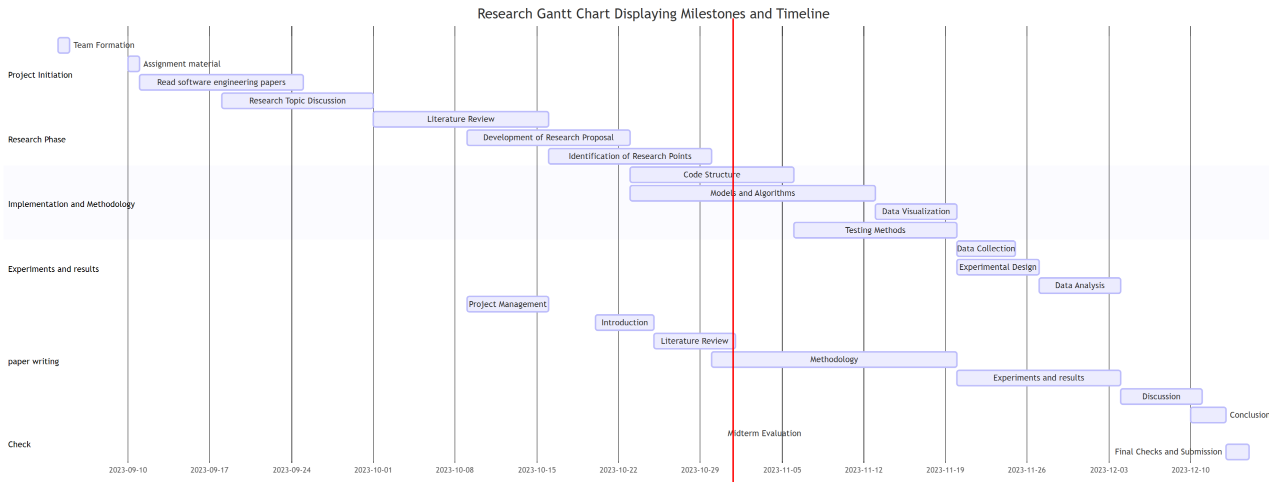

Fig. 1. Research Gantt Chart Displaying Milestones and Timeline

fundamentally equal and characterized by active engagement. Below is a detailed breakdown of the tasks and responsibilities that each team member undertook:

- Jinpeng Zhai
  - Actively engaged in the implementation of the research plan, transforming initial concepts into concrete project components.
  - Took the lead in formulating and developing the research methodology, playing a crucial role in crafting the research approach presented in the paper.
  - Contributed significantly to the writing of the research's methodology and approach sections in the paper, ensuring its thoroughness and accuracy.
- Ziqi Yang
  - Demonstrated proactiveness in conducting extensive research to identify available tools and the latest research methods.
  - Presented innovative ideas for implementation and actively participated in the design and development phases of the research.
  - Contributed to the creation of project components and the development of research solutions, enriching the research's technical foundation.
- Yiran Zhao
  - Formed and efficiently managed the research team, facilitating regular team meetings and fostering effective communication.
  - Contributed to the research paper, including creating the overleaf template and drafting the introduction section and project management section.
  - Thoroughly reviewed relevant literature, bringing a wealth of knowledge and insights to the research. Introduced innovative ideas and approaches that enriched the research process and contributed to the research's overall success.

The entire team is in unanimous agreement that each member's commitment and contributions to the research project were equitable and integral to its success. The collaborative and complementary efforts of all team members were essential in achieving the project's goals.

D. Future Research Directions

References

[1] A. Saabith, M. Fareez, and T. Vinothraj, "Python current trend applications-an overview," *International Journal of Advance Engineering and Research Development*, vol. 6, no. 10, 2019.

[2] Z. Zhang, H. Zhu, M. Wen, Y. Tao, Y. Liu, and Y. Xiong, "How do python framework apis evolve? an exploratory study," in *2020 ieee 27th international conference on software analysis, evolution and reengineering (saner)*. IEEE, 2020, pp. 81–92.

[3] Y. Peng, Y. Zhang, and M. Hu, "An empirical study for common language features used in python projects," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021, pp. 24–35.

[4] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in github," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, 2014, pp. 155–165.

[5] B. Åkerblom, J. Stendahl, M. Tumlin, and T. Wrigstad, "Tracing dynamic features in python programs," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 292–295.

[6] S. Cao, Y. Zeng, S. Yang, and S. Cao, "Research on python data visualization technology," in *Journal of Physics: Conference Series*, vol. 1757, no. 1. IOP Publishing, 2021, p. 012122.

[7] A. Fernández Blanco, "Empirical foundation for memory usage analysis through software visualizations," 2023.

[8] "cpython/lib/trace.py at main · python/cpython," GitHub. [Online]. Available: https://github.com/python/cpython/blob/main/Lib/trace.py

[9] "trace — trace or track python statement execution — python 3.10.4 documentation," docs.python.org. [Online]. Available: https://docs.python.org/3/library/trace.html