Reflection and Traceability Report on EcoOptimizer

Ayushi Amin
Tanveer Brar
Nivetha Kuruparan
Sevhena Walker
Mya Hussain

We largely followed our development plan regarding team meetings, communication, member roles, and workflow. We adhered to our planned meeting schedule, ensuring regular discussions to track progress, resolve blockers, and align our work with project goals. To keep everything organized, all types of meetings

were documented as issues on GitHub. This allowed us to track progress effectively, link discussions to specific tasks, and maintain meeting notes for future reference.

Our communication plan also worked well—whether through scheduled check-ins or ad-hoc discussions, we maintained a steady flow of information via our chosen platforms. Each team member upheld their assigned roles, ensuring a balanced distribution of tasks. Our workflow remained structured, with clear milestones and responsibilities that kept the project on track. While there were some natural adjustments along the way to optimize efficiency, we remained aligned with the overall plan.

Regarding technology, we successfully used the tools and frameworks outlined in our development plan. Our refactoring library was developed in Python, leveraging Rope for refactoring, PyLint for inefficient code pattern detection, and Code Carbon for energy analysis. For code quality, we enforced PEP 8 standards and used Ruff for linting and Pyright for static type checking. Additionally, we incorporated PySmells for detecting code smells.

To ensure robust testing, we wrote unit tests using pytest, integrated with coverage.py to measure code coverage. We also implemented performance benchmarking using Python's built-in benchmarking tools to measure execution time across different file sizes.

For version control and CI/CD, we relied on GitHub and GitHub Actions, streamlining our development process through automated testing and integration. Our VS Code plugin was built using TypeScript, aligning with the VS Code architecture.

Overall, we effectively used the planned technologies and tools, making only necessary refinements to optimize development.

6.2 What Went Well?

One of the biggest strengths was how we stayed organized. We used GitHub Issues to track tasks, discussions, and even meeting notes, which made it easy to monitor progress and ensure accountability. This approach helped keep everything transparent and well-documented.

Communication was another strong point. We stuck to our planned check-ins but also had the flexibility to reach out whenever needed. This balance kept things moving without feeling rigid. We also made sure to follow PEP 8 coding standards and used linters, which kept our code consistent and easy to read.

The tools we used made a big difference in keeping things smooth. GitHub Actions handled our automated testing and integration, so we didn't have to worry about manually running tests every time. PyTest and Coverage.py helped us stay on top of testing, while Ruff and PyLint made sure our code was clean and error-free.

For the actual project, Code Carbon gave us energy consumption insights (even if it wasn't always perfectly accurate), and Rope made refactoring much easier, saving us a lot of time.

Overall, the combination of good teamwork, clear processes, and the right tools kept us organized and made development a lot more efficient.

6.3 What Went Wrong?

One of the main challenges we faced in the project was moving away from using reinforcement learning. Initially, it seemed like a great way to optimize energy consumption, but as we decided not to use it, the direction of the project had to shift. This required us to adjust our approach, which took time and created some uncertainty within the team. Keeping everyone aligned and maintaining clear communication was essential during this transition, especially as we explored alternative methods to achieve our goals.

On the technology side, developing the refactoring library itself turned out to be more complicated than we initially anticipated. Creating a tool that could not only identify energy-saving opportunities but also refactor the code while preserving its intent was a delicate balance. Working with Python's unique performance and dynamic features added another layer of complexity. Despite these challenges, they provided valuable learning experiences that have shaped how we approach the project now.

6.4 What Would you Do Differently Next Time?

For our next project, one thing we'd do differently is spend more time upfront validating the technical approach before diving into development. With the shift away from reinforcement learning, we had to make adjustments mid-way through, which slowed down progress. In the future, we'd prioritize a more thorough exploration of different technologies and approaches early on to avoid these kinds of pivots. We'd also make sure to keep a closer eye on scope to prevent unnecessary shifts that could affect the timeline.

Additionally, we'd focus on improving team coordination from the start, ensuring that everyone is aligned not just on the technical goals but also on the project's overall direction. Regular, structured check-ins would be helpful to track progress and address roadblocks quickly. On the technical side, we would invest more time in setting up a robust DevOps pipeline earlier in the process, ensuring that continuous integration and testing are seamless from the beginning. Overall, we believe these changes would help streamline both the technical and team dynamics for a smoother project experience.

7 Reflection on Capstone

7.1 Which Courses Were Relevant

Several courses we've taken were highly relevant to our capstone project:

- **Software Architecture**: This course gave us a solid foundation in designing scalable, maintainable, and efficient software systems, which was

essential when building the refactoring library and ensuring the overall structure of our tool was optimal.

- **Data Structures and Algorithms**: The principles from this course helped us design more efficient algorithms for analyzing and refactoring source code to optimize energy consumption. Understanding how to work with data structures effectively was key in handling different code patterns and optimization techniques.
- **Database Systems**: Although we didn't directly deal with complex databases in the capstone, understanding how to efficiently manage and query large datasets was useful when we considered tracking energy usage metrics or managing configurations within the system.
- **Real-Time Systems and Control Applications**: This course provided insights into managing time-sensitive tasks and the real-time performance of systems, which was helpful when considering the energy optimization of software execution, particularly in the context of how different coding choices could impact performance in real-time.
- **Intro to Software Development**: This course was crucial for learning best practices in documentation and understanding design patterns, which helped us structure our code and communicate our approach effectively throughout the project.
- **Object-Oriented Programming**: The concepts from this course were directly applied when designing our system, especially when managing the relationships between the different components of our refactoring library.
- **Human Computer Interfaces**: This course helped us a lot with usability testing and designing the frontend of our tool in VS Code, ensuring that it was user-friendly and intuitive for anyone using the software.
- **Software Engineering Practice and Experience: Binding Theory to Practice**: This course was instrumental in teaching us how to approach open-ended design problems and apply both theoretical and practical knowledge to real-world scenarios. It was particularly helpful in guiding us through the experiential approach to solving computational problems, especially when considering embedded systems and assembly programming.

These courses equipped us with the necessary technical background to approach the project's challenges, from system design to algorithm optimization, and they directly informed the decisions we made while building the energy optimization tool.

7.2 Knowledge/Skills Outside of Courses

For our capstone project, we had to acquire several skills and knowledge areas that were not directly covered in our coursework:

- **Energy-Efficient Software Development**: We had to research and understand techniques for reducing energy consumption in software, including best practices for writing energy-efficient code and tools for measuring energy usage.
- **Static Code Analysis**: Since our project involved analyzing and refactoring code, we had to learn about static analysis techniques and how to extract meaningful insights from source code without executing it.
- **Refactoring Strategies for Energy Optimization**: While we had learned about code refactoring in some courses, optimizing for energy efficiency was a new challenge. We had to explore strategies that improve performance while minimizing power consumption.
- **GitHub DevOps and CI/CD Pipelines**: Although we had experience with version control, setting up automated testing and continuous integration workflows in GitHub required additional learning.
- **VS Code Extension Development**: Since our tool was designed to work within VS Code, we had to learn how to develop and integrate extensions, which was not covered in our coursework.
- **User Research and Usability Testing**: While our Human-Computer Interfaces course covered usability principles, we had to go deeper into conducting user research, gathering feedback, and refining our tool's user experience.
- **Performance Profiling and Benchmarking**: Measuring the energy impact of different coding techniques required us to explore profiling tools and benchmarking methods to ensure our refactorings were actually improving efficiency.
- **Advanced Python Optimization Techniques**: Since our refactoring library is Python-based, we needed to learn about Python-specific optimizations, including memory management, just-in-time compilation techniques, and efficient data structures.

These additional skills allowed us to successfully design and implement our energy optimization tool, bridging the gap between our academic knowledge and the real-world challenges of software efficiency.

Table 1: Feedback Responses for MIS, MG, Development Plan

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
TA (Issue #515)	MG/MIS not formalized	• Added math formalizations in MG for different refactorers that lent well to a discrete math implementaiton.	Issue #515
TA (Issue #513)	MIS needs abstraction	• Removed Python-specific code • Abstracted common functions	Commits: 46fbcc9, d1a4506
TA (Issue #512)	Spelling inconsistencies	• Updated to Canadian English • Fixed table references	PR #529, PR #530
TA (Issue #511)	Diagram orientation issues	• Redesigned hierarchy diagrams • Standardized downward flow	PR #530
TA (Issue #510)	Module secret overlap	• Consolidated	PR #530