Reflection and Traceability Report on EcoOptimizer

Ayushi Amin
Tanveer Brar
Nivetha Kuruparan
Sevhena Walker
Mya Hussain

1 Changes in Response to Feedback

1.1 SRS and Hazard Analysis

1.1.1 Software Requirement Specification Document (SRS)

The updated SRS shows a significant shift in technical direction, most notably through the **complete removal of the reinforcement learning model** that was previously central to the architecture. This change cascaded through multiple sections—the business data model was simplified by eliminating the learning component, related business use cases were deleted, and all associated requirements about algorithmic refactoring optimization were stripped out.

Conversely, substantial new functionality was added around **IDE plugin configuration and user control**. The document now includes detailed requirements for customizable smell displays, selective refactoring of specific issues, and toggleable linting features. These are supported by new product use cases that outline workflows for filtering smells and clearing analysis history.

The integration with **GitHub Actions was completely excised** from the requirements, removing all CI/CD pipeline functionality that appeared in the original version. Security requirements were simultaneously simplified by eliminating authentication needs and privacy compliance mandates.

New visualization elements were incorporated, including a **comprehensive state diagram** that maps the complete refactoring workflow. The code smell taxonomy was refined to focus on more concrete, measurable issues while removing vaguer categories. Throughout the document, qualitative performance targets were replaced with quantified metrics.

These changes collectively reflect an evolution from experimental concepts to a more focused tool, with stronger emphasis on **practical IDE integration** and user-configurable analysis.

Feedback Description	Source	Implementation	Issue/Commit Links
Add priority planning to phase in plan	peer	duplicates and discarded	Issue #136 + #155
Clarify meaning of FR-8	peer	addressed	Issue #137
Clarify definition of "lowest energy consumption"	peer	addressed	Issue #138

Continued on next page

Table 1 – continued from previous page

Feedback Description	Source	Implementation	Issue/Commit Links
Specify which software developers are considered to be shareholders and how their feedback will be used	peer	addressed	Issue #140
Explain why training requirements aren't needed for the project	peer	addressed	Issue #141
Expand more of the energy consumption dashboard	peer	addressed	Issue #152
Fix labelling for immunity requirements	peer	addressed	Issue #153
Add measurable indicators of success to security fit criterion	peer	discarded	Issue #153
Clarify amiguous FR-2	peer	discarded	Issue #154
Create entity relationship diagram	peer	discarded	Issue #156
Add adaptability requirements	peer	addressed	Issue #157
Add symbolic constants	TA	addressed	Issue #269
Move glossary up in the document	TA	addressed	Issue #270
Reference tables and figures	TA	addressed	Issue #271
Make requirements more abstract	TA	addressed	Issue #272
Add traceability	TA	addressed	Issue #273
Make requirements less ambiguous	TA	addressed	Issue #274
Formalize document	TA	addressed	Issue #275

Table 1: SRS Feedback Tracking and Implementation

1.1.2 Hazard Analysis

The revised Hazard Analysis reflects a **significant simplification and refocusing of the system's scope**, with particular attention to **user experience, security, and measurable requirements**. Most notably, the team **eliminated the Reinforcement Learning Model** entirely—a major departure from the original design. This removal streamlined the document, excising associated hazards (like model overfitting and bias) and requirements (such as quarterly model audits), while allowing greater focus on core refactoring functionality. The **Testing Module was similarly consolidated** into the Analysis and Refactoring Modules, with validation now implicitly handled through smell detection and performance metrics rather than as a standalone component.

The energy measurement backend shifted from **PyJoules to CodeCarbon**, with updated failure miti-

gations (e.g., cloud-environment permissions in HZ-3) and revised assumptions about cross-platform consistency. The new version also introduces **robust UI/accessibility safeguards**, adding five previously absent hazards—such as real-time metric display (HZ-12) and WCAG compliance (HZ-15)—backed by quantified requirements like `METRIC_REFRESH_TIME = 2 sec`. Security saw deliberate hardening, with expanded static analysis to **prevent refactoring of sensitive code** (HZ-8) and detect external library dependencies (HZ-7).

Peripherals (GitHub Action and Web Client), were replaced with a **strictly local VS Code extension**, emphasizing tighter integration and reducing external dependencies. New features like **checkpointing (SCR-10)**—enforcing progress saves every 30 seconds—and **concurrency controls (SCR-9)** address crash recovery and parallel operation risks, underscored by symbolic constants (`SAVE_TIME`, `MAX_DATA_LOSS`) for precision. Together, these changes reflect a **more pragmatic, user-centric tool** with clearer boundaries, measurable safeguards, and mitigated risks—all while preserving the original goal of energy-efficient Python refactoring.

Feedback Description	**Source**	**Implementation**	**Issue/Commit Links**
Clarify the terminology used in SCR 9	peer	addressed	Issue #185
Fix referencing between hazards and requirements	peer	addressed	Issue #186
Add new hazard related to unexpected system shutdown	peer	addressed	Issue #187
Fix clashing hazards	peer	addressed, but component completely removed in later design iteration	Issue #189
Fix FMEA table formatting	peer and TA	addressed	Issue #190, Issue #278
Critical assumptions section lacks user and environment context assumptions	peer	addressed	Issue #191
FMEA table missing analysis of inter-component hazards and failure modes	peer	addressed	Issue #192
Unclear recommended actions point for reinforcement learning component	peer	addressed, but component removed in later design iteration	Issue #193
Unclear critical assumptions	peer	addressed	Issue #194
Need symbolic constants	TA	addressed	Issue #277
Make clear which components are in scope	TA	addressed, but removed in later iteration	Issue #279

Table 2: Hazard Analysis Feedback Tracking and Implementation

1.2 MG, MIS and Development PLan

The following table details all changes made to the MIS, MG and Development Plan.

Notable improvements include the formalization of discrete mathematical models for refactoring operations (Issue #515), abstraction of core optimization logic to work across programming languages logic (Issue #513), and consolidation of AST security modules (Issue #510). Additional enhancements focused on documentation rigor through Canadian English standardization (Issue #512), LaTeX quote correction (Issue #265), and explicit risk mitigation mapping (Issue #267). These changes collectively strengthened the system's theoretical foundations while improving team accountability through quantified performance metrics (Issue #266) and role definitions (Issue #264).

Table 3: 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 implementation	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 AST-related secrets	PR #530
TA (Issue #267)	Risk identification in PoC	• Added explicit risk section • Mapped mitigations	9218c22
TA (Issue #266)	Vague team charter	• Defined quantitative metrics	663fcd3

Continued on next page

Table 3 continued from previous page

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
TA (Issue #265)	Incorrect LaTeX quotes	• Replaced all "quotes" • Added linter rule	2572497
TA (Issue #264)	Undefined team roles	• Assigned specific responsibilities • Created role descriptions	663fcd3

1.3 VnV Plan and Report

something

2 Challenge Level and Extras

2.1 Challenge Level

2.2 Extras

3 Design Iteration (LO11 (PrototypeIterate))

4 Design Decisions (LO12)

5 Economic Considerations (LO23)

5.1 Market Viability

There is a growing market for sustainability-focused developer tools, driven by corporate Environmental, Social, and Governance (ESG) goals and rising cloud computing costs. EcoOptimizer targets the 16.7 million active Visual Studio Code users [**?**], particularly enterprise developers and organizations seeking to reduce energy consumption in code deployment. Competitors like code linters and performance optimizers exist, but EcoOptimizer uniquely ties refactoring to CO2 reduction metrics, aligning with global net-zero initiatives.

5.2 Marketing Strategy

To promote EcoOptimizer, we propose:

- **Community Engagement**: Launch on developer forums (Reddit, HackerNews, Stack Overflow) and open-source platforms (GitHub).
- **Partnerships**: Collaborate with cloud providers (AWS, Google Cloud).
- **Content Marketing**: Publish case studies demonstrating energy cost savings (e.g., "Reducing AWS bills by 12% via loop optimization").
- **Freemium Model**: Offer basic optimizations for free, with premium features (custom CO2 metrics, CI/CD integration) for paid tiers.

5.3 Production Costs

Developing a commercially viable version of EcoOptimizer requires strategic allocation of resources across three key areas:

- **Further Development**: We would allocate $20,000 to further the development of the tool and add more useful features and better security. The $20,000 development cost reflects industry-standard freelance rates for full-stack Python/TypeScript development. At $20/hour [**?**], this covers 1,000 hours of work.
- **Maintenance**: Monthly costs of $1,000 are driven primarily by cloud hosting. Using Google Cloud's pricing calculator [**?**], we estimate $500/month for backend servers and $300/month for API usage, with $200 reserved for incremental updates.
- **Marketing**: We would allocate an initial $5,000 for marketing that could change as more users join or time progresses. The $5,000 upfront investment aligns with HubSpot's benchmarks for bootstrapped developer tools [**?**], covering SEO optimization, paid ads targeting VS Code users, and content creation for technical blogs. As we are just starting out we dont have a ton of money to dump into marketing from the start.

Total first-year operational costs amount to $37,000, positioning EcoOptimizer competitively against similar sustainability tools with higher upfront R&D budgets.

5.4 Pricing and Profitability

EcoOptimizer adopts a **freemium model** to maximize adoption while ensuring sustainability, offering:

- **Free Tier**: Basic code optimization (loop simplification, dead code removal) to build trust and community engagement.
- **Individual Tier**: Priced at $5/month or $50/year, this *paid* tier unlocks advanced features like CI/CD integration and custom CO2 dashboards. Aligns with premium VS Code extensions like *CodeGPT* [**?**]. To break even on the $37,000 first-year operational costs, EcoOptimizer requires **740 annual subscribers** ($37,000 / $50 per user). Given VS Code's 14M-user base [**?**], this represents just 0.0053% of the total market—a feasible target within 12 months.
- **Enterprise Tier**: $1,000/year per-team licenses include priority support and multi-repo analysis, reflecting premium ESG tool pricing in the current market.

Focusing solely on the Individual Tier, EcoOptimizer achieves break-even by acquiring **740 annual subscribers** ($37,000 / $50 per user). This requires converting just **0.0053%** of VS Code's 14M users, a highly attainable target given industry benchmarks for niche developer tools.

6 Reflection on Project Management (LO24)

6.1 How Does Your Project Management Compare to Your Development Plan

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.