Reflection and Traceability Report on EcoOptimizer

Ayushi Amin
Tanveer Brar
Nivetha Kuruparan
Sevhena Walker
Mya Hussain

Contents

1 Changes in Response to Feedback

1.1 SRS and Hazard Analysis

1.1.1 Software Requirement Specification Document (SRS)

The SRS underwent significant refinements based on feedback, beginning with peer suggestions to clarify ambiguous requirements including FR-2 (Issue #154) and FR-8 (Issue #137), with particular attention given to properly defining "lowest energy consumption" (Issue #138) to ensure measurable outcomes. The documentation around stakeholders was enhanced to explicitly identify which software developers would be considered shareholders (Issue #140), while training requirement exclusions were properly justified (Issue #141). Additional details expanded the energy consumption dashboard description (Issue #152), and new adaptability requirements were added (Issue #157). Priority planning suggestions were reviewed but ultimately discarded as duplicates (Issues #136 + #155).

Teaching assistant feedback drove structural improvements including moving the glossary section (Issue #270) and implementing symbolic constants (Issue #269). All tables and figures were properly referenced throughout (Issue #271), while requirements were made more abstract (Issue #272) with improved traceability between sections (Issue #273). The team formalized the document by standardizing terminology (Issue #274) and reducing ambiguity in requirement wording (Issue #275). Immunity requirement labeling corrections (Issue #153) and security metric additions were implemented, though some suggestions like creating an entity relationship diagram were ultimately discarded (Issue #156).

Table 1: SRS Feedback Tracking and Implementation

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
peer	Add priority planning to phase in plan	None, None, discarded	Issue #136 + #155
peer	Clarify meaning of FR-8	addressed in issue	Issue #137
peer	Clarify definition of "lowest energy consumption"	addressed in issue	Issue #138
peer	Specify which software developers are considered to be shareholders and how their feedback will be used	addressed in issue	Issue #140
peer	Explain why training requirements aren't needed for the project	addressed in issue	Issue #141
peer	Expand more of the energy consumption dashboard	addressed in issue	Issue #152
peer	Fix labelling for immunity requirements	addressed in issue	Issue #153

Continued on next page

Table 1 continued from previous page

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
peer	Add measurable indicators of success to security fit criterion	None, discarded	Issue #153
peer	Clarify ambiguous FR-2	None, discarded	Issue #154
peer	Create entity relationship diagram	None, discarded	Issue #156
peer	Add adaptability requirements	addressed in issue	Issue #157
TA	Add symbolic constants	addressed in issue	Issue #269
TA	Move glossary up in the document	addressed in issue	Issue #270
TA	Reference tables and figures	addressed in issue	Issue #271
TA	Make requirements more abstract	addressed in issue	Issue #272
TA	Add traceability	addressed in issue	Issue #273
TA	Make requirements less ambiguous	addressed in issue	Issue #274
TA	Formalize document	addressed in issue	Issue #275

1.1.2 Hazard Analysis

The hazard analysis documentation was substantially improved through multiple feedback iterations. Terminology in SCR-9 was clarified (Issue #185) and hazard/requirement references were corrected (Issue #186). A new unexpected system shutdown hazard was added (Issue #187), while clashing hazards were addressed before some components were later removed (Issue #189). Both peer and TA feedback improved FMEA table formatting (Issue #190 + Issue #278), and the critical assumptions section gained user/environment context (Issue #191). Additional analysis covered inter-component hazards (Issue #192), and reinforcement learning component actions were clarified before its removal (Issue #193). Unclear assumptions were resolved (Issue #194), while TA feedback ensured proper symbolic constant usage (Issue #277) and component scope clarification (Issue #279), though some analyzed components were later excluded during design evolution.

Table 2: Hazard Analysis Feedback Tracking and Implementation

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
peer	Clarify the terminology used in SCR 9	addressed	Issue #185

Continued on next page

Table 2 continued from previous page

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
peer	Fix referencing between hazards and requirements	addressed	Issue #186
peer	Add new hazard related to unexpected system shutdown	addressed	Issue #187
peer	Fix clashing hazards	• Addressed initially • Component completely removed in later design iteration	Issue #189
peer and TA	Fix FMEA table formatting	addressed	Issue #190, Issue #278
peer	Critical assumptions section lacks user and environment context assumptions	addressed	Issue #191
peer	FMEA table missing analysis of inter-component hazards and failure modes	addressed	Issue #192
peer	Unclear recommended actions point for reinforcement learning component	• Addressed initially • Component removed in later design iteration	Issue #193
peer	Unclear critical assumptions	addressed	Issue #194
TA	Need symbolic constants	addressed	Issue #277
TA	Make clear which components are in scope	• Addressed initially • Components removed in later iteration	Issue #279

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 rigour 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
TA (Issue #265)	Incorrect LaTeX quotes	• Replaced all "quotes" • Added linter rule	2572497

Continued on next page

Table 3 continued from previous page

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
TA (Issue #264)	Undefined team roles	• Assigned specific responsibilities • Created role descriptions	663fcd3

1.3 VnV Plan and Report

1.3.1 VnV Plan

The VnV Plan underwent significant revisions based on feedback from both TAs and peers, with all addressed feedback items documented in Table 4. The most substantial changes involved restructuring our test plans to align with evolving requirements in the SRS (see Section 1.1 for detailed SRS modifications). As the project requirements were refined, we systematically removed obsolete test cases that no longer matched the current specifications and modified existing tests to better verify the updated functional and non-functional requirements. Particular attention was given to ensuring traceability between test cases and requirements through a revised traceability matrix. The automated testing strategy was also enhanced to accommodate new validation scenarios while maintaining comprehensive coverage of the system's core functionality.

More information on changes made to VnV Plan can be found in this issue where commits are linked.

Table 4: Feedback Responses for VnV Plan

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
Peer (Issue #222)	Add development plan to the list of relevant documentation section	• No changes were made. We did not reference the development plan in the vnv plan.	N/A
Peer (Issue #223)	Clarification on Usability Survey Question	• Surveys were updated for the usability testing session and were moved to the Extras folder.	e53b27c
Peer (Issue #224)	Manual vs Automatic Test Control	• Reviewed all tests in plan to make sure whether they are automatic or manual	4c5acf7
Peer (Issue #225)	Inconsistent Formatting in Test Requirements Section	• Reviewed Test Plan and fixed all formatting	4c5acf7

Continued on next page

Table 4 continued from previous page

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
Peer (Issue #226)	Unclear definition for unauthorized external modifications	• Added clarity to test-SRT-3	d71ed8d
Peer (Issue #227)	Clearer acceptance criteria for maintainable and adaptable codebase	• More clarification to these terms in provided in the Process. Will not fix.	N/A
Peer (Issue #238)	Inconsistent Fields Used for Requirements	• It didn't really make sense for static tests to share the same testing template as dynamic ones • Modified the template slightly so that all tests begin with the type field so that it is more clear that the test is static or dynamic	6069c72
TA (Issue #505)	Additional Citations Needed	• Added more citations	392d2fe
TA (Issue #506)	Fix Spelling and Format	• Re-read the documentation to fix	c882090
TA (Issue #507)	State How Survey Data will be Used	• Added more clarification and details on how the survey data will be used in testing	653e7a3
TA (Issue #508)	Need More Quantifiable Metrics for Usability	• Added more metrics for usability in related tests	653e7a3

1.3.2 VnV Report

We updated the VnV Report to address all the feedback from peers and TAs (see Table 5), but that wasn't the only reason for the changes. Since the VnV Plan had evolved—especially with new tests and coverage metrics—we needed to sync the report with those updates. This meant adding results from the newly implemented tests, adjusting the traceability matrix, and expanding our analysis to reflect the current testing approach.

Beyond just feedback fixes, we also made general improvements to the report. These included better organization of test results, clearer explanations of our methodology, and updates to match the project's current

state. Many of these unrelated tweaks were tracked in Issue #531, which served as our central hub for report revisions.

The biggest changes landed in Sections 7 and 8 (now with proper system/non-functional test coverage) and Section 9 (where we added detailed code coverage analysis). Throughout the report, we focused on keeping everything consistent with both the updated VnV Plan and the project's actual progress.

Table 5: Feedback Responses for VnV Report

Feedback Source	**Feedback Summary**	**Changes Made**	**Related Commits/Issues**
Peer (Issue #485)	Trace to Requirements/Modules	• Duplicate issue of Issue #424 and Issue #423 • We have separate issues that contain these details	N/A
Peer (Issue #486)	Functional Requirements Evaluation Clarity	• Report contains numerous tables that clearly indicate which tests passed or failed for each subsection, and these tables are formatted for visual clarity. No change needed	N/A
Peer (Issue #487)	Hyperlinking to Previous Reports with the term "Code Smell"	• Code smell is a generic programming term. Added reference to the descriptions for code smells in the section.	1970f9d
Peer (Issue #488)	Definition of Each Code Smell	• Added explanation for each code smell	23a69c7
Peer (Issue #489)	Missing Table Captions	• Added descriptive captions	3786f19
Peer (Issue #490)	Partial in Feedback and Implementation Plan	• Completed all test feedback	cf39eb7

Continued on next page

Table 5 continued from previous page

Feedback Source	Feedback Summary	Changes Made	Related Commits/Issues
Peer (Issue #492)	Functional requirement; specific syntax errors introduced in the test file are not mentioned	• The Purpose of this test is to verify the system's ability to detect and handle any Python syntax error, not specific error types. Therefore, there is no need to specify the exact syntax error.	N/A
Peer (Issue #493)	Clear names for sample code smell	• Added clarification on extensibility	2118fa0
Peer (Issue #494)	Section 9: Code Coverage Metrics	• No longer relevant as tests were implemented/updated and new coverage was updated in the report	41c2c83
Peer (Issue #496)	Refer users to SRS and VnV Plan	• Added cross-reference section • Included document links	6989094
TA (Issue #578)	Some tests incomplete due to features	• Marked pending tests and completed them	2349ee1
TA (Issue #577)	Spelling, grammar and style	• Proofread entire document	88f1f5d

2 Challenge Level and Extras

2.1 Challenge Level

The challenge level for this project is **general**, as agreed upon with the course instructor.
This project required integrating multiple technical components, including developing a usable VS Code extension from scratch and a Python library to detect, measure and refactor energy-inefficient code patterns. While these aspects were non-trivial, the scope and complexity aligned with a general challenge level rather than advanced.

2.2 Extras

Two extras were completed as part of this project:

- **User Onboarding Guide:** A detailed onboarding guide was developed and made available via the project wiki. It includes:

  - Setup and installation instructions.
  - Explanation of plugin features and functionality.
  - Sample workflows for using the plugin.
  - Embedded video demonstrations to assist users visually.

  The onboarding guide can be accessed here.

- **Usability Testing:** Usability testing was conducted to evaluate the effectiveness and user-friendliness of the plugin. The testing process included:

  - Scenario-based user walkthroughs.
  - Survey responses measuring ease of use and satisfaction.
  - Identification of usability issues and areas for improvement.

  A full report of the usability testing is available here.

3 Design Iteration (LO11 (PrototypeIterate))

The design of our project underwent several significant iterations throughout its development, each driven by technical considerations, stakeholder feedback, and usability testing results. This evolution reflects our team's commitment to creating a practical solution that effectively addresses user needs while remaining technically feasible within project constraints.

Initial Project Vision
The project began as a Python library focused on code optimization, with supplementary components including an IDE plugin, web-based metrics visualization and a GitHub Actions that could fully integrate our library into a workflow as secondary features. Our original technical approach emphasized reinforcement learning (RL) as the core optimization mechanism, based on its potential for adaptive code improvement.

Pivotal Technical Reassessment
In January, a crucial meeting with our project supervisor prompted a fundamental redesign. After consultation with one of our supervisor's graduate students who provided both project feedback and a reinforcement learning primer, we collectively determined that:

- The RL approach would be set aside in favour of rule-based refactoring
- The IDE plugin would transition from peripheral feature to primary interface

This strategic shift was motivated by several factors:

- The complexity and time requirements of implementing effective RL exceeded our project timeline
- Rule-based refactoring offered more predictable, explainable results
- Emphasizing the plugin interface would better serve our target users (developers) in their natural workflow environment

Visual Studio Code was selected as our target IDE early in the process due to its widespread adoption and robust extension capabilities.

Scope Refinement After Revision 1
Following our first formal review with course professors, we made the deliberate decision to remove features like a GitHub Actions and web-based metrics from our scope. This simplification allowed us to concentrate our efforts on refining functionality we already had while maintaining a realistic development timeline.

Usability-Driven Interface Improvements
In early March, we conducted two structured usability testing sessions that yielded valuable feedback (detailed here). These sessions revealed several key pain points and opportunities for enhancement:

- **Interface Clarity**: Users struggled with sidebar visibility and button distinction, leading us to:
  - Overhaul the sidebar completely. We went from a sidebar that only populated when a refactoring was in session to a permanent fixture that was an integral part of using accessing the extension's features.
  - Add features like constant smell detection.
- **User Guidance**: New users required more onboarding support, resulting in:
  - Development of step-by-step instructions manual
  - Addition of progress indicators to manage expectations during refactoring
- **Customization Needs**: Users requested more personalization options, prompting us to:
  - Implement colour customization for different smell types
  - Expand explanatory content about energy-saving benefits

Final Implementation Approach
The current implementation reflects these iterative improvements while respecting project constraints. We prioritized changes that addressed:

- High-frustration pain points (e.g., button visibility)
- Critical usability gaps (e.g., lack of user guidance)
- Valuable customization options
- Clear communication of system status

This evolutionary process demonstrates our user-centred design philosophy, where technical decisions were continually evaluated against both implementation feasibility and real user needs. The result is a more focused, usable tool that maintains its core value proposition while offering an improved user experience.

4 Design Decisions (LO12)

The implementation of EcoOptimizer was shaped by technical limitations, project constraints, and core design principles. The system was built to be modular, extensible, and energy-aware, while remaining usable and performant in a local development environment.

Architectural Decisions

- **Refactorer Design Structure:** The refactoring subsystem follows a two-level inheritance hierarchy:
  - `BaseRefactorer` defines the interface and shared logic for all refactorers.
  - `MultiFileRefactorer` extends `BaseRefactorer` to support transformations across multiple files.
  - Concrete refactorers subclass either `BaseRefactorer` or `MultiFileRefactorer`, depending on their scope.

  This structure enforces separation of concerns, encourages reusability, and makes it easy to introduce new refactorers without modifying the core system.

Limitations

- **Tooling Limitations for Code Smell Detection:** No single tool could detect all required smells. A strategy-based analyzer system was introduced to combine AST, Astroid, Pylint, and custom analyzers dynamically.
- **Energy Measurement Accuracy:** Energy readings from CodeCarbon vary with hardware and OS. To reduce variability, measurements are performed in isolated subprocesses and compared under consistent conditions.
- **File-Level Focus:** While most refactorings are file-scoped, limited multi-file analysis was included for smells requiring cross-file changes.
- **Temporary File Refactoring:** All transformations are staged in temporary directories before applying to source files. This protects code integrity but adds complexity to file handling and rollback.

Assumptions

- **Valid Python Input:** The system assumes syntactically correct Python files as input to avoid expensive pre-validation routines.
- **Code Executability:** Energy measurement assumes the input file can be run as a standalone Python script using a main entry point.
- **Library Availability:** It is assumed that CodeCarbon and its dependencies are installed and operational across user systems.

Constraints

- **Performance Trade-offs:** Analyzer and refactorer execution was parallelized and subprocessed to preserve responsiveness, especially on large codebases.
- **Cross-Platform File Management:** Temporary directories and subprocess logic were adapted for compatibility across Windows (`TEMP`) and Unix-based systems (`TMPDIR`).
- **Security and Stability:** Refactoring operations and energy measurements are isolated to prevent modifications to user files before validation. Logging and error handling ensure graceful degradation on failures.
- **Ecosystem Fit:** Libraries like FastAPI, Pydantic, and CodeCarbon were selected based on open-source licensing and compatibility with modern Python development practices.

Overall, the architecture of EcoOptimizer balances security, performance and extensibility. Each design decision reflects a trade-off that prioritizes energy transparency and safe refactoring, while remaining mindful of usability and future expandability.

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 [Eyal Katz(2025)], 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 [Upwork(2025)], 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 [Google Cloud(2025)], 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 [HubSpot(2025)], 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* [codeGPT(2025)]. 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 [Eyal Katz(2025)], 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.

References

[codeGPT(2025)] codeGPT. Codegpt pricing, 2025. URL https://codegpt.co/pricing. Accessed: 2025-04-04.

[Eyal Katz(2025)] Eyal Katz. Top 20 best vscode extensions for 2025, 2025. URL https://www.jit.io/blog/vscode-extensions-for-2023. Accessed: 2025-04-04.

[Google Cloud(2025)] Google Cloud. Pricing calculator, 2025. URL https://cloud.google.com/products/calculator. Accessed: 2025-04-04.

[HubSpot(2025)] HubSpot. Startup marketing cost benchmarks, 2025. URL https://www.hubspot.com/marketing-statistics. Accessed: 2025-04-04.

[Upwork(2025)] Upwork. Freelancer rates guide, 2025. URL https://www.upwork.com/hire/software-developers/cost/. Accessed: 2025-04-04.