Software Requirements Specification for Software Engineering: subtitle describing software

Team 4, EcoOptimizers

Nivetha Kuruparan
Sevhena Walker
Tanveer Brar
Mya Hussain
Ayushi Amin

March 24, 2025

Contents

Revision History

Date	Name	Notes
October 11th, 2024	All	Created initial revision of SRS
January 2nd, 2025	All	Added clarification to training requirements
January 5th, 2025	All	Update FR-7, FR-8, Ideas for Solution
January 6th, 2025	All	Fixed link references to tables and figures and added preambles to those sections, move glossary section, added symbolic constants
February 8th, 2025	All	Removed requirement for interfacing with GitHub Actions.
February 10th, 2025	All	Updated the data dictionary and business data model
February 10th, 2025	All	Updated costs, compliance requirements, security requirements and scope of the product.
March 24th, 2025	All	Updated and added functional requirements.
March 24th, 2025	Mya Hussain	Updated Usability and Humanity Requirements.
March 24th, 2025	Ayushi Amin	Updated maintainability and support requirements.
March 24th, 2025	Ayushi Amin	Updated cultural requirements.

Symbolic Constants

Name	Value
ENERGY_SAVE	5%
SMELL_COVERAGE	80%
TEST_FUNCTION_THRESH	100%
REFACTOR_EFFICACY_THRESH	95%
REFACTOR_REVERT_LIMIT	5 commits
CRITICAL_ENERGY_SAVE_THRESH	10%
MIN_USER_CONFIDENCE	70%
MAX_TASK_CLICKS	4 clicks
MIN_USER_EOU	80%
SMALL_FILE_TIME	5 sec
LARGE_FILE_TIME	30 sec
REFACTOR_TIME	10 sec
DETECTION_ACC	90%
LARGE_CODE_BASE_TIME	2 min
NEW_REFACTOR_TIME	7 days
COMPREHENSION_TIME	2 days
ROLLBACK_TIME	1h
MIN_CODE_COVERAGE	80%
OS_PERF_DIFF_LIMIT	5%

Table 1: Table of Symbolic Constants

1 Purpose of the Project

1.1 User Business

The Information and Communications Technology (ICT) sector, an essential component of the global economy, is responsible for 2-4% of global CO2 emissions today, with projections suggesting this could rise to 14% by 2040 (?). To meet sustainability goals, including a 72% reduction in CO2 emissions by 2040 (?), this sector must find ways to improve energy efficiency.

One area of concern is the energy consumption of software systems. However, for software engineers, it is not practical for them to focus on optimizing energy consumption while developing complex programs. Instead, supporting tools and technologies are needed to assist in improving energy efficiency without altering the intended behaviour of the software.

This project aims to tackle Python, a popular but energy inefficient programming language. Python consumes significantly more energy compared to more efficient languages like C and Rust—over 70 times more energy, on average, for similar tasks (?). The project's goal is to develop tools to reduce Python's energy consumption through automated refactoring. While this will not solve the entire problem of the carbon footprint associated with the software, it is a step towards more energy-efficient practices in the software development process.

1.2 Goals of the Project

Purpose: The purpose of this project is to provide software engineers with tools to optimize the energy efficiency of Python programs by automating refactoring suggestions, while still allowing users to review and decide whether to apply the changes.
Advantage: By reducing the energy consumption of Python programs, this project will contribute to the broader effort of decreasing the carbon footprint of software development, supporting the ICT sector's goal of reducing CO2 emissions by 72% by 2040 (?).
Measurement: The project's success can be measured by the reduction in energy usage achieved after applying the refactorings. Benchmarks will compare the energy consumption of original and refactored code, to achieve a measurable percentage reduction in energy consumption.

2 Naming Conventions and Terminology

2.1 Glossary of All Terms, Including Acronyms, Used by Stakeholders involved in the Project

supervisor A member of faculty from McMaster University responsible for overseeing a project being worked on by students taking the SFWRENG 4G06 Capstone course.

large-scale applications Applications that manage high volumes of data, users, or transactions, typically requiring scalable architectures.

cloud-hosted applications Software applications deployed and run on remote cloud servers, accessible over the internet.

environmental footprint The total impact an activity or product has on the environment, measured by metrics like carbon emissions and resource consumption.

refactoring The process of restructuring existing code without changing its external behaviour to improve readability, performance, or maintainability.

mobile environment A software environment specifically designed for mobile devices, such as smartphones or tablets, which have limited resources.

embedded environment A software environment where applications run on specialized hardware with constrained resources, often without traditional operating systems.

SaaS Software as a Service (SaaS) refers to cloud-based software applications delivered to users via the internet on a subscription basis.

backend The part of a software system that handles server-side logic, database interactions, and application functionality not directly visible to users.

software developer A professional who designs, writes, and maintains software applications or systems.

data analyst A professional who processes and analyzes large sets of data to help organizations make informed decisions.

tech company A company focused on technology products and services, including software, hardware, and IT services.

freelance Self-employed individuals who offer specialized services, such as software development, without long-term commitments to any employer.

usability testing A method of evaluating how easy and user-friendly a software application or product is by observing real users interacting with it.

Python Code Refers to the original Python code given by the end user to refactor.

Refactored Code Refers to the Python code that had refactorings made to it.

library A collection of pre-written code that developers can reuse in their projects to add specific functionalities.

Git A distributed version control system that allows multiple developers to track changes in source code, collaborate, and manage project history efficiently.

GitHub A web-based platform used for version control and collaboration on code through Git repositories.

Actions A GitHub feature that automates tasks such as testing, deployment, and continuous integration via custom workflows.

workflow A sequence of automated steps or actions that define a process, often for continuous integration, deployment, or testing.

Visual Studio Code A free, open-source code editor developed by Microsoft, known for its versatility and wide range of extensions.

VS Code An abbreviation of Visual Studio Code

Visual Studio Code (VS Code) marketplace An online platform where users can discover, install, and manage extensions that enhance the functionality of Visual Studio Code.

JSON JavaScript Object Notation, a lightweight data format used to store and exchange information between systems.

XML Extensible Markup Language, a data format used to encode documents in a way that is both human-readable and machine-readable.

package manager A tool that automates the process of installing, updating, and managing software packages or libraries in a project.

PIP A package manager for Python that simplifies the installation and management of Python libraries.

IDE Integrated Development Environment, a software application providing tools like a code editor, debugger, and compiler to facilitate development.

progress indicators Visual or textual cues, such as loading bars or percentages, that inform users about the status of ongoing processes.

plugin A software component that adds specific features or functionalities to an existing software system.

configuration file A file used to define settings or preferences for a software application, often stored in a human-readable format like JSON or XML.

dashboard A user interface that provides an overview of key information and metrics, typically presented in graphs, charts, and tables.

sync The process of ensuring that data is consistent across multiple systems or devices by automatically updating changes in real-time.

programming language A formal language used to write software programs by providing instructions that a computer can execute.

Python A programming language known for its simplicity and versatility, widely used in web development, data science, and automation.

Java A programming language known for its portability and scalability, commonly used for enterprise-level applications.

C/C++ A programming language family used for system programming, game development, and applications requiring high performance.

C# A programming language developed by Microsoft, primarily used for developing applications on the .NET platform.

JavaScript A programming language used primarily for adding interactivity to web pages and building dynamic web applications.

TypeScript A programming language that builds on JavaScript by adding static typing, improving code reliability and scalability.

Go A programming language created by Google, designed for simplicity and efficiency in building scalable applications.

Rust A programming language focused on safety, performance, and concurrency, often used in system programming.

3 Stakeholders

The stakeholders involved in this project include all individuals and groups that have a direct or indirect interest in the development, implementation, and usage of the refactoring library for energy efficiency. These stakeholders influence project decisions and will be impacted by their outcomes. Understanding their roles and expectations is crucial for ensuring that the library meets the needs of its users and aligns with broader organizational goals.

This section introduces the **client**, **customer**, and **other stakeholders** that are involved in the project. Finally, we will look at the users of the product, specifically the **hands-on-users**, what their **personas** may look like, their **priority** levels, and what kind of **participation** we can expect from them throughout the development of this product.

3.1 Client

The client of this project is **Dr. Istvan David** from McMaster's Department of Computing and Software. As the project supervisor, his role is to guide the development team with his technical and domain expertise. As the client, he sets the product's requirements and will be involved throughout its development.

3.2 Customer

The customers of this product are **software developers**. Specifically, they are the developers that work in teams for small to large corporations, or freelancers looking to improve their services. They will be the primary users of the product and, therefore, will offer critical feedback on its effectiveness such as suggestions for improvement and/or additional features. Any feedback received from this stakeholder will be given top priority for consideration as the goal for this tool is to be an integral part of a software developer's workflow.

3.3 Other Stakeholders

Project Managers

They oversee project operations and focus on reducing energy costs associated with large-scale or cloud-hosted applications. They might leverage the refactoring library to reduce operational costs and achieve business sustainability goals.

Business Sustainability Teams

This stakeholder is responsible for reducing the company's environmental footprint by analyzing its energy emissions. They will use the energy efficiency metrics provided by the refactoring library to improve environmental sustainability practices within their organization.

End Users

End users refer to the users of software that uses the product in its development. They will indirectly reap benefits from these applications that have been optimized using the refactoring library. Software used, especially mobile or embedded environments where battery life is a key concern, might prove to be more responsive and efficient. They have no involvement in the development of the product.

Regulatory Bodies

This stakeholder is responsible for establishing regulations governing energy consumption and sustainability standards. They can promote the adoption of energy-efficient software practices and potentially certify tools that meet regulatory standards.

3.4 Hands-On Users of the Project

Software Developers

- **User Role**: Integrate library into the codebase, provide tests to check refactoring against original functionality
- **Subject Matter Experience**: Journeyman to Master
- **Technological Experience**: Journeyman to Master
- **Attitude toward technology**: Varies (conservative to positive)
- **Physical location**: Remote (at home), in-person (work office) or hybrid

Business Sustainability Teams

- **User Role**: Access metrics provided by library
- **Subject Matter Experience**: Journeyman
- **Technological Experience**: Novice to Journeyman
- **Attitude toward technology**: Neutral to positive

3.5 Personas

Persona: Raven Reyes

Age: 37
Job Title: Senior Software Developer
Education: Bachelor's in Computer Science

Work Environment: Works at a mid-sized SaaS company with a focus on improving their environmental footprint.
Professional Background: Has over 15 years of experience in software development, specializing in backend systems. Worked with various programming languages (Python, Java, and C++), and is well-versed in optimizing code for performance.

Need: With the company more focused on sustainability, Raven and her team need to go through their codebase and apply energy efficient changes to their code.
Challenges: Knowing what to change in their code to make it more efficient is challenging, not to mention the incredible amount of code they will have to sift through. We are talking hundreds of thousands of lines of code!

Persona: Christopher Robin

Age: 34
Job Title: Data Analyst
Education: Bachelor's in Data Science

Work Environment: Works at a large corporation that has recently started increasing their efforts to become a sustainable company.
Professional Background: Over 8 years of experience in data analysis, just recently looking into sustainability metrics. Christopher regularly collaborates with IT teams to track performance metrics and recently energy consumption metrics to help identify areas for improvement.

Need: Christopher needs accurate data on energy consumption from the company's software systems to generate insights that drive sustainability initiatives and help meet corporate environmental targets.
Challenges: Translating raw technical data into meaningful insights that can guide decisions is difficult, especially when working with complex systems.

Persona: Draco Malfoy

Age: 29
Job Title: Freelance Software Developer
Education: Bachelor's in Software Engineering

Work Environment: Works remotely on multiple freelance projects for small and mid-sized businesses, focusing on web applications and backend systems.
Professional Background: Has 7 years of experience working with various clients to develop and optimize software, primarily in Python and JavaScript. He often works on tight deadlines, where balancing performance and development speed is key.

Need: Draco needs efficient ways to optimize the code he writes for clients, particularly in terms of performance and energy efficiency, as more businesses become environmentally conscious.
Challenges: As a freelancer, time is money. Draco needs tools that help him quickly identify inefficiencies in the code and refactor them, without spending hours analyzing large codebases or learning new systems.

3.6 Priorities Assigned to Users

Key Users: Software Developers, Business Sustainability Teams
Secondary User: Project Managers

3.7 User Participation

For the bulk of the development process, requirements will be gathered from the development team itself with the help of the project supervisor, Dr. Istvan David.

During the testing phase, usability testing will be conducted to further refine the product.

3.8 Maintenance Users and Service Technicians

Due to the nature of this project as a capstone requirement, there are currently no expected maintenance users.

4 Mandated Constraints

4.1 Solution Constraints

MD-SL 1. *The system must refactor Python code without changing its original functionality.*

Rationale: The project is focused on energy-efficient refactoring for Python code, and altering functionality could break existing software behaviour.
Fit Criterion: The system must pass all original test cases after refactoring, ensuring that functionality remains unchanged.
Priority: High

MD-SL 2. *The refactored code must result in measurable energy savings.*

Rationale: The project's primary objective is to improve the code's energy efficiency. Refactorings should lead to a noticeable reduction in energy consumption.
Fit Criterion: The system must show at least a ENERGY_SAVE reduction in energy consumption, as measured by tools like `CodeCarbon`, for most of the refactored code.
Priority: High

MD-SL 3. *The refactored code must be provided to the user upon completion of the refactoring process.*

Rationale: Developers need access to the refactored code to utilize it in their projects immediately after the refactoring process.
Fit Criterion: The system must deliver the refactored code to the user in a clear format, ensuring it is readily available for implementation.
Priority: High

4.2 Implementation Environment of the Current System

MD-EC 1. *The product shall be able to run on standard laptop environments, including typical developer setups with operating systems such as Windows, macOS, and Linux.*

Rationale: Developers will primarily use the tool on personal workstations, including laptops, and it must integrate smoothly with typical development environments. Supporting standard laptop environments ensures that the tool is accessible to a wide range of users without the need for specialized hardware.
Fit Criterion: The tool must be installable and functional on a standard laptop with Visual Studio Code. It should perform well without requiring excessive processing power or memory.
Priority: High

4.3 Partner or Collaborative Applications

The project will focus on developing the tool with flexibility for integration with Visual Studio Code. While no specific partner applications are required, understanding these possibilities can aid smoother integration. Future collaboration with Python development tools may be considered, but no formal interface constraints are needed at this stage.

4.4 Off-the-Shelf Software

The project has no strict requirements for off-the-shelf software but will leverage open-source libraries to enhance functionality and maintain flexibility. Tools like CodeCarbon for energy measurement and Python analysis libraries may be used. While no legal issues are expected, all tools will be assessed for compatibility. Documentation will be maintained, though no specific constraints are set at this stage.

4.5 Anticipated Workplace Environment

The workplace will include standard software development setups. The tool should offer a non-intrusive interface suited for quiet workspaces and providing quick feedback in collaborative environments.

4.6 Schedule Constraints

SCHD 1. *The project shall be completed by April 2025, with interim deadlines for key milestones such as Proof of Concept (November 2024) and the final demonstration (March 2025).*

Rationale: These deadlines are based on the academic timeline and the expectations of the capstone course.
Fit Criterion: All project components must be completed and fully functional by the final demonstration in March 2025.
Priority: High

4.7 Budget Constraints

BDGT 1. *The project shall not exceed the resources available to the team, which includes free open-source software and free services like GitHub for hosting.*

Rationale: The team does not have a budget for paid services or proprietary software.
Fit Criterion: The project must be implemented using free tools and libraries and hosted on GitHub.
Priority: High

4.8 Enterprise Constraints

ENTP 1. *The product shall be built to comply with the standards of McMaster University's capstone project requirements and academic integrity policies.*

Rationale: The project is part of the university's curriculum and must adhere to its standards.
Fit Criterion: The product must meet the requirements specified by the course syllabus and project advisor.
Priority: High

5 Relevant Facts And Assumptions

5.1 Relevant Facts

Not applicable to this system.

5.2 Business Rules

The following are some business rules established between the team.

1. **Project Timeline Adherence:** All milestones must be completed according to the established project timeline. Any delays or unexpected circumstances must be reported to the team as soon as possible.

2. **Pull Request Review Requirement:** All pull requests must receive at least two independent reviews before they can be merged into the main branch. Reviewers must provide feedback or approval within 48 hours of the request to ensure timely progress.

3. **Team Communication Standard:** All team members are required to communicate in a friendly and respectful manner during discussions, meetings, and in all written communications. Constructive feedback should be provided with the intent to support and enhance team collaboration.

5.3 Assumptions

It is assumed for this system that users will be seeking to refactor Python code and will make use of Visual Studio Code.

6 The Scope of the Work

This section defines the boundaries and objectives of the work, focusing on the tasks and components required to develop and deliver the refactoring library. It provides a clear view of how the system operates within its context and breaks the work into logical partitions to facilitate development and implementation.

6.1 The Current Situation

The current software development landscape often prioritizes functionality and performance over energy efficiency. Many existing Python codebases are not optimized for energy consumption, leading to unnecessary power usage and increased carbon footprint. The following aspects characterize the current situation:

1. **Manual Refactoring:** Developers typically perform refactoring manually, which is time-consuming and prone to errors.
2. **Limited Awareness:** Many developers lack awareness of energy-efficient coding practices and their impact on overall energy consumption.
3. **Absence of Automated Tools:** There is a lack of widely adopted automated tools specifically designed to refactor Python code for energy efficiency.
4. **Performance-Centric Optimization:** Most existing optimization tools focus on performance improvements rather than energy efficiency.
5. **Inefficient Code Patterns:** Many codebases contain inefficient code patterns that consume more energy than necessary.

The project aims to address these issues by:

1. **Automated Refactoring Library:** Developing a Python library that automatically detects and refactors code for improved energy efficiency.
2. **IDE Integration:** Creating a Visual Studio Code plugin that integrates the refactoring library, providing real-time suggestions and automated refactoring options.
3. **Energy Consumption Measurement:** Implementing tools to measure and compare energy consumption before and after refactoring.
4. **Developer Education:** Raising awareness about energy-efficient coding practices through the tool's suggestions and documentation.
5. **Continuous Improvement:** Implementing a reinforcement learning model to improve refactoring suggestions over time based on developer feedback and real-world energy savings data.

6.2 The Context of the Work

The purpose of this subsection is to illustrate the flow of inputs, outputs, and interactions between the refactoring library and its external systems, such as developers, energy measurement tools, and machine learning frameworks. Figure 1 highlights the connections between the system's components and external elements, ensuring a comprehensive understanding of its operational environment.

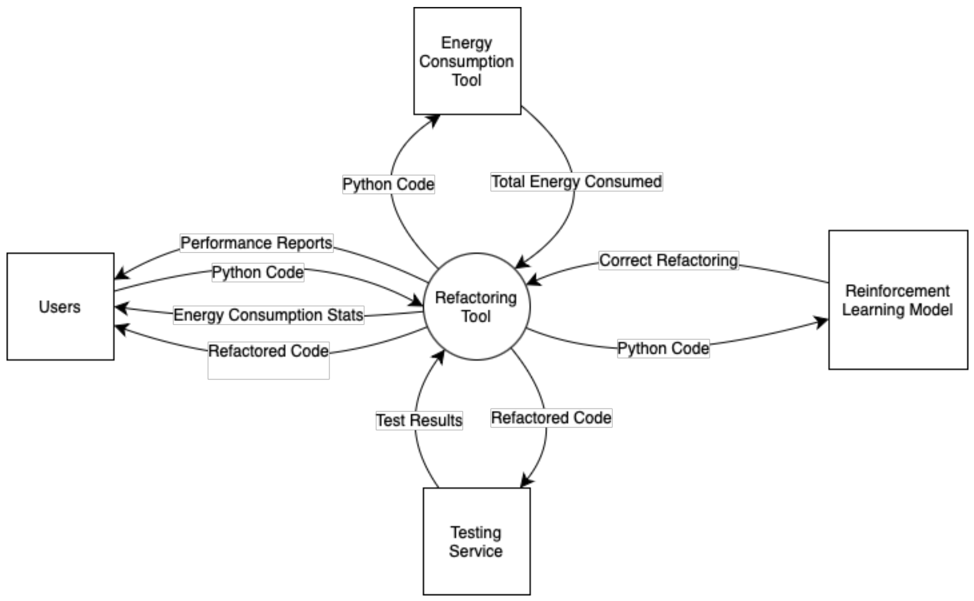

Figure 1: Work Context Diagram

6.3 Work Partitioning

In this subsection, the work needed to complete this system is divided into distinct activities, such as identifying code smells, applying refactorings, and measuring energy efficiency. As seen in Table 2, each partition outlines its purpose, dependencies, and deliverables to provide a structured overview of the project's tasks.

Event #	Event Name	Input	Output(s)
1	Users submit Python code	Python Code	Refactored Code
2	Energy Analysis of code	Python Code	Total Energy Consumed
3	RI Model produces refactoring	Python Code	Correct Refactoring
4	Testing and Validation of refactored code	Refactored Code	Test Results
5	Reporting Performance metrics of new code	Refactored Code	Performance Reports
6	Viewing Energy consumption reports	Refactored Code	Energy Consumption Statistics

Table 2: Work Partitioning of System

6.4 Specifying a Business Use Case (BUC)

Each event listed in Table 2 is expanded into an individual business use case which describes how the system handles specific scenarios.

BUC 1: Code Submission

Input: Python Code
Output: Refactored Code
Pre-condition: User uses either GitHub Action or a VS Code plugin to submit code to the refactoring tool

Scenario:

1. Refactoring tool receives the Python code
2. PyJoules is used to store energy consumption data for the original Python code submitted
3. Tool analyzes the code for inefficiencies (PySmells)

4. Python code is provided to the Re-enforcement learning model to find a refactoring
5. Energy consumption is measured of refactored code and compared to the original data
6. Refactored code is tested to ensure functionality is maintained from the original code
7. Refactored code is received by the user

Sub Variation:

- *4a:* If no PySmells are identified, then code is returned to the user
- *6a:* If energy consumption increases for refactored code, the reinforcement model is asked to find another refactoring
- *7a:* If code functionality is not preserved for refactored code, the reinforcement model is asked to find another refactoring

BUC 2: Energy Analysis of Code

Input: Python Code
Output: Total Energy Consumed
Pre-condition: Submission of Python code to the Energy Consumption Tool

Scenario:

1. Tool receives Python Code
2. Energy consumed is measured during execution
3. The analysis results are compiled into a report
4. Report of total energy consumed is received by the refactoring tool

BUC 3: Reinforcement Learning Model Produces Refactoring

Input: Python Code
Output: Correct Refactoring
Pre-condition: Request for refactored code from the Reinforcement Learning Model

Scenario:

1. Model receives Python Code
2. Analyze Code for potential refactoring
3. Generate suggestions based on previous learning and data
4. Implement suggested refactorings

Sub Variation:

- *2a:* If there are no refactorings found, Model outputs are given code back to the refactoring tool

BUC 4: Testing and Validation of Refactored Code

Input: Refactored Code
Output: Test Results
Pre-condition: Energy consumed for refactored code is less than the energy consumed for original code

Scenario:

1. Conduct tests on refactored code
2. Conduct tests on original code
3. Validate results of refactored code to the results of the original code to ensure functionality is intact
4. Signal to refactoring tool to send refactored code to the user

Sub Variation:

- *4a:* If functionality is not preserved, signal to the refactoring tool to refactor again

BUC 5: Reporting Performance Metrics of New Code

Input: Refactored Code
Output: Performance Reports
Pre-condition: Testing and validation is completed successfully

Scenario:

1. Generate detailed performance report based on testing outcomes
2. User receives the performance report

BUC 6: Viewing Energy Consumption Reports

Input: Refactored Code
Output: Energy Consumption Statistics
Pre-condition: Testing and validation is completed successfully

Scenario:

1. Comprehensive statistics are compiled from energy analysis data
2. Information is compiled in an accessible format for developers to review

7 Business Data Model and Data Dictionary

This section describes the structure and organization of the data that flows through the refactoring library. It explains how the system's components interact with data entities, ensuring a consistent and well-defined understanding of the information processed by the system.

7.1 Business Data Model

The following diagram (Figure 2) illustrates the relationships between key components of the system as well as their interactions with external components.

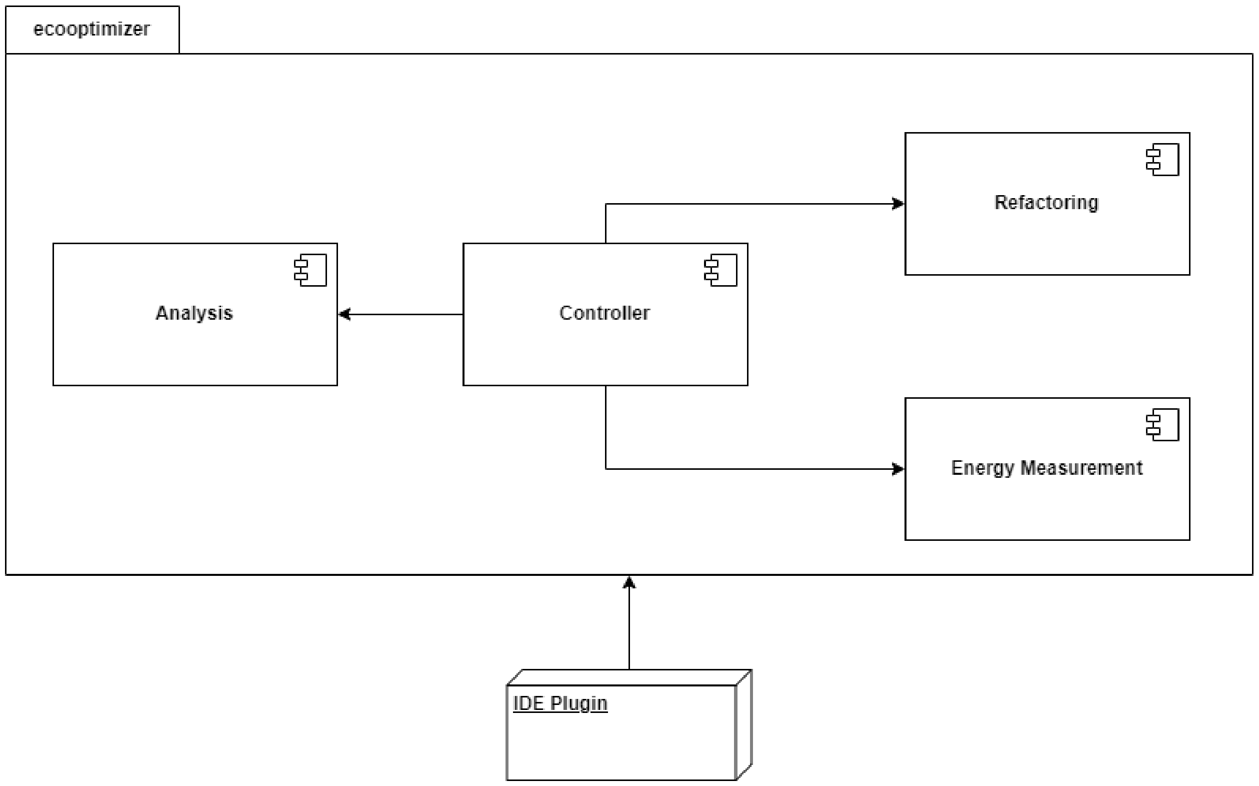

Figure 2: Business Data Model of System

7.2 Data Dictionary

Table 3 shown below defines each component in the system, including its attributes, type, and purpose. It ensures clarity and consistency in data handling and serves as a reference for development and testing.

Name	**Content**	**Type**
ecooptimizer	Controller + Analysis + Refactoring + Energy Measurement	package
IDE extension	A plugin containing the `ecooptimizer` package delivered as an IDE extension	External Service Provider
Controller	Controlls the flow of execution within the package. Responsible for calling other modules and handling outputs	Module
Refactoring	Contains all necessary tools for refactoring the code smells defined in the package	Module
Energy Measurement	measures the energy consumption of the given source code	Module
Analysis	Contains all necessary tools for the analysis of the code smells defined in the package	Module

Table 3: Data Dictionary for the System

8 The Scope of the Product

This section outlines the boundaries and functionality of the refactoring library, detailing what the system will and will not deliver. It focuses on the internal components of the library, their interactions, and how they collectively address the project's objectives.

8.1 Product Boundary

This subsection includes a diagram (Figure 3) showing the system's boundary, identifying its internal components and their interactions. It clarifies what falls within the scope of the product and what lies outside its responsibility.

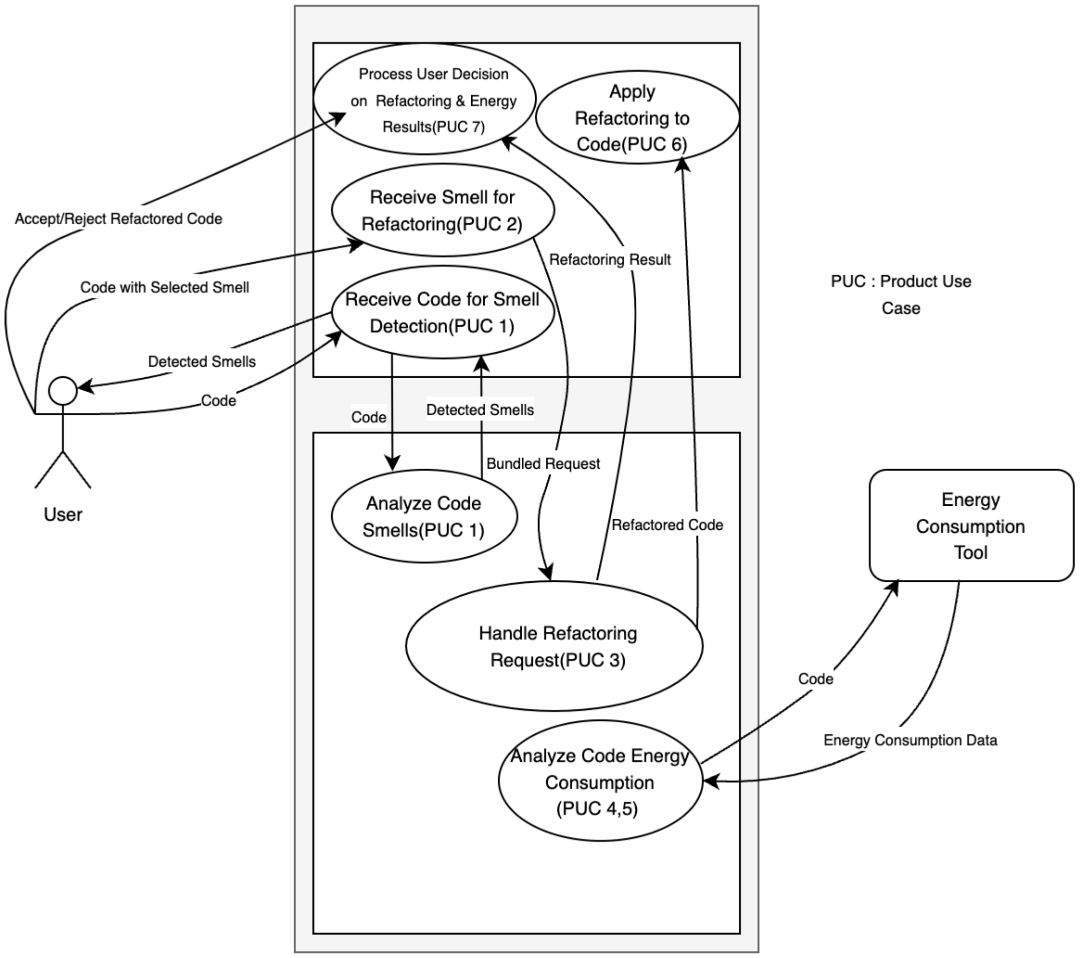
Process User Decision on Refactoring & Energy Results(PUC 7)
Apply Refactoring to Code(PUC 6)
Receive Smell for Refactoring(PUC 2)
Receive Code for Smell Detection(PUC 1)
Accept/Reject Refactored Code
Code with Selected Smell
Detected Smells
Code
User
Refactoring Result
PUC : Product Use Case
Code
Detected Smells
Bundled Request
Analyze Code Smells(PUC 1)
Refactored Code
Handle Refactoring Request(PUC 3)
Energy Consumption Tool
Code
Energy Consumption Data
Analyze Code Energy Consumption (PUC 4,5)

Figure 3: Product Boundary Diagram of System

8.2 Product Use Case Table

The following table summarizes the primary use cases of the refactoring library, such as identifying code smells, measuring energy consumption, applying refactorings, and allowing users to accept or reject refactored code. Each use case briefly describes the interaction between the user and the system.

PUC #	PUC Name	Actor/s	Input & Output(s)
1	Detect Code Smells	VS Code Plugin (via Refactoring Lib.)	Original Code (in), Identified Code Smells (out)
2	Toggle Smell Linting	User	Toggle Command (in), Enabled/Disabled Linting (out)
3	Filter Code Smells	User	Filter Criteria (in), Filtered Smells (out)
4	Configure Smell Detection	User	Threshold Parameters (in), Updated Detection Rules (out)
5	Select Code Smell	User	Selected Code Smell (in)
6	Refactor Code	VS Code Plugin (via Refactoring Lib.)	Code Segment (in), Refactored Code (out)
7	Measure Energy (Before)	Refactoring Library (via Energy Tool)	Original Code (in), Initial Energy Results (out)
8	Measure Energy (After)	Refactoring Library (via Energy Tool)	Refactored Code (in), Final Energy Results (out)
9	Show Results	VS Code Plugin	Refactored Code, Performance Metrics (out)
10	Accept/Reject Results	User	Refactored Code, Energy Results (in), Decision (out)
11	Clear Smell History	User	Clear Command (in), Reset Cache (out)
12	Access Documentation	User	Help Request (in), Documentation (out)
13	Generate System Logs	System	Log Request (in), Log File (out)

Table 4: Product Use Case Table of System

8.3 Individual Product Use Cases (PUC's)

This subsection expands on the use cases listed in Table 4, providing a detailed description of each. It explains how the system components work together to fulfill each use case, emphasizing expected functionality and outcomes for users.

PUC 1: Detect Code Smells

Trigger: The VS Code Plugin processes the input code using the refactoring library.

Preconditions:

- The VS Code Plugin has received the original code.
- The Refactoring Library is active and ready to analyze code.

Actors: VS Code Plugin (via Refactoring Library).
Outcome: Code smells are identified and presented to the user.
Input: Original Python Code.
Output: List of detected code smells.

PUC 2: Select Code Smell for Refactoring

Trigger: The user selects a detected code smell to be refactored.

Preconditions:

- The system has identified code smells.
- The user is reviewing the detected smells.

Actors: User.
Outcome: The selected smell is sent to the refactoring tool.
Input: Selected Code Smell.
Output: Selected smell is marked for refactoring.

PUC 3: Refactor Code

Trigger: The user has selected a code smell for refactoring.

Preconditions:

- Code smells have been detected.
- The user has selected a smell to refactor.

Actors: VS Code Plugin (via Refactoring Library).
Outcome: The system refactors the selected code smell.
Input: Code Segment.
Output: Refactored Python Code.

PUC 4: Measure Energy Consumption Before Refactoring

Trigger: The refactoring library submits the original code to the energy measurement tool.

Preconditions:

- The system has received the original code.
- The energy measurement tool is active and connected.

Actors: Refactoring Library (via Energy Measurement Tool).
Outcome: Initial energy consumption is measured.
Input: Original Python Code.
Output: Initial Energy Consumption Results.

PUC 5: Measure Energy Consumption After Refactoring

Trigger: The refactored code is submitted to the energy measurement tool.

Preconditions:

- The system has refactored the code.
- The energy measurement tool is active and connected.

Actors: Refactoring Library (via Energy Measurement Tool).
Outcome: Final energy consumption is measured.
Input: Refactored Python Code.
Output: Final Energy Consumption Results.

PUC 6: Show Results

Trigger: Refactored code and energy consumption metrics are ready.

Preconditions:

- The refactoring process has completed.
- Energy consumption before and after refactoring has been measured.

Actors: VS Code Plugin.
Outcome: The refactored code and performance metrics are presented to the user.
Output: Refactored Code, Energy Consumption Results.

PUC 7: Accept or Reject Refactoring Results

Trigger: The user reviews the refactored code and its energy consumption results.

Preconditions:

- The system has presented the refactored code and energy consumption results.

Actors: User.
Outcome: The user accepts or rejects the refactored code. If rejected, no changes are applied. If accepted, the refactored code replaces the original.
Input: Refactored Code, Energy Consumption Results.
Output: Accepted or Rejected Decision.

9 Functional Requirements

9.1 Functional Requirements

FR 1. *The system must accept Python source code files.*

Rationale: The system needs to process Python code as its primary input to refactor and improve energy efficiency.
Fit Criterion: The system successfully processes valid Python files without errors and provides feedback for invalid files.
Priority: High

FR 2. *The system must identify specific code smells that can be targeted for energy saving.*

Rationale: Energy inefficiencies are often related to well-known code smells, so identifying them is the first step in improving efficiency.
Fit Criterion: The tool should be meet the SMELL_COVERAGE for the detection and reporting of the following smells: Long Parameter List (LPL), Long Message Chain (LMC), Long Element Chain (LEC), Long Lambda Function (LLF), Complex List Comprehension (CLC), Member Ignoring Method (MIM), Cache Repeated Calls (CRC), String Concat in Loop (SCL).
Priority: High

FR 3. *The system must suggest at least one appropriate refactoring for each detected code smell to decrease energy consumption or indicate that none can be found.*

Rationale: For developers to optimize their code, the tool must provide an appropriate refactoring suggestion based on detected code smells.
Fit Criterion: The suggested refactored code demonstrates a measurable improvement in energy consumption as measured in kg.
Priority: High

FR 4. *The system must produce valid refactored Python code as output or indicate that no possible refactorings were found.*

Rationale: Refactored code must remain functional and error-free to ensure maintainability and usability.
Fit Criterion: The output code is syntactically correct and adheres to Python standards, validated by an automatic linter.
Priority: High

FR 5. *The tool must require Python version 3.10 to run but must be capable of analyzing and refactoring Python code written for versions 3.8 and newer.*

Rationale: The tool leverages features available in Python 3.10 for its operation while ensuring compatibility with analyzing codebases written in Python 3.8 and 3.9, which are the most widely adopted recent major versions in use.
Fit Criterion: The tool operates correctly in a Python 3.10 environment and successfully analyzes and refactors code written for Python versions 3.8, 3.9, and newer.
Priority: Medium

FR 6. *The system must generate and display energy consumption metrics.*

Rationale: Developers need clear metrics as a justification for energy efficient refactorings.

Fit Criterion: Energy consumption metrics are clear, well-structured, and provide actionable insights, allowing users to easily understand the results.
Priority: Medium

FR 7. *The tool must provide comprehensive documentation and help resources.*

Rationale: Detailed documentation is necessary to help users install, understand, and use the tool effectively.
Fit Criterion: Documentation covers installation, usage, and troubleshooting, receiving positive feedback for clarity and completeness from users.
Priority: Medium

FR 8. *The system shall provide developers with refactoring suggestions within an IDE before changing code, allowing them to review and approve energy-efficient changes.*

Rationale: Giving developers control over which refactorings are applied ensures that they can maintain the balance between energy efficiency and their coding style or project requirements.
Fit Criterion: The IDE plugin must display at least two refactoring options for inefficient code patterns, allowing developers to either apply or reject them before making any changes.
Priority: Medium

FR 9. *The system shall provide developers with refactoring suggestions within an IDE before changing code, allowing them to review and approve energy-efficient changes.*

Rationale: Giving developers control over which refactorings are applied ensures that they can maintain the balance between energy efficiency and their coding style or project requirements.
Fit Criterion: The IDE plugin must display at least two refactoring options for inefficient code patterns, allowing developers to either apply or reject them before making any changes.
Priority: Medium

FR 10. *The system must allow users to filter detected code smells within the IDE plugin.*

Rationale: Developers should be able to focus on specific types of inefficiencies based on their project needs and priorities.
Fit Criterion: The plugin provides a user-friendly interface that enables filtering detected code smells by category, severity, or type, ensuring a customized refactoring experience.
Priority: Medium

FR 11. *The system should indicate code smells within a smell-linted file.*

Rationale: Developers need a clear visual representation of detected code smells to efficiently identify problematic areas in their code.
Fit Criterion: The system highlights detected code smells within the source file, for example, by underlining or marking affected lines.
Priority: High

FR 12. *The system must allow for refactoring of specific smells.*

Rationale: Developers should have the flexibility to refactor only selected code smells rather than applying all refactorings at once.
Fit Criterion: Users can click on a specific detected smell and choose to refactor only that issue.
Priority: High

FR 13. *The system must allow users to configure specific smell detection parameters.*

Rationale: Different projects may have varying definitions of what constitutes a long lambda function or a long message chain, so customizable thresholds improve flexibility.
Fit Criterion: The system provides a configuration interface where users can set thresholds for smell detection (e.g., defining the maximum acceptable length for a lambda function or the number of chained method calls before triggering a warning).
Priority: Medium

FR 14. *The system should generate and allow users to access logs of system processes.*

Rationale: Logs provide transparency on system operations, helping users debug issues and track changes.
Fit Criterion: The system generates logs detailing detected smells, applied refactorings, and system errors, accessible through an interface or log file.
Priority: Medium

FR 15. *The system must allow users to accept or reject suggested refactored changes and apply the corresponding action.*

Rationale: Developers should have control over the changes made to their code to ensure refactorings align with project requirements.
Fit Criterion: The tool provides an interface where users can review, accept, or reject suggested refactorings before they are applied.
Priority: High

FR 16. *The system must allow users to toggle smell linting on and off within the IDE.*

Rationale: Users should be able to control whether code smell detection is actively running, especially to avoid distractions in certain workflows.
Fit Criterion: The system provides a toggle button to turn smell linting on or off. When this is enabled, the tool automatically highlights smells in open files, and when disabled, all highlighting/indications are removed.
Priority: Medium

FR 17. *The system must allow users to remove the history of detected smells and other cached data.*

Rationale: Clearing stored data helps users maintain a clean workspace, manage storage, and reset the tool for a fresh start.
Fit Criterion: The system provides an option to delete previously detected smells

and cached data, ensuring a reset state when needed.
Priority: Medium

FR 18. *The system should allow for the enabling and disabling of specific smells for detection.*

Rationale: Developers may want to focus on certain types of inefficiencies while ignoring others based on project-specific needs.
Fit Criterion: The system provides an interface where users can enable or disable specific code smells from being detected and reported.
Priority: Medium

10 Look and Feel Requirements

10.1 Appearance Requirements

LFR-AP 1. *The IDE plugin refactoring interface shall present the original and refactored code side by side, allowing developers to compare and choose between them easily.*

Rationale: Providing a side-by-side view of the original and refactored code helps developers make informed decisions about applying changes.
Fit Criterion: The interface must display the original code on one side and the refactored code on the other, with clear options for developers to accept or reject the refactorings without confusion.
Priority: High

LFR-AP 2. *The tool shall have a minimalist design, focusing only on essential elements to reduce clutter.*

Rationale: A clean and simple interface allows developers to focus on the code and refactoring suggestions without distractions, improving usability.
Fit Criterion: The tool should prominently display only the code, refactoring suggestions, and energy metrics, omitting unnecessary visual elements or distractions.
Priority: Low

10.2 Style Requirements

LFR-ST 1. *The tool shall convey a professional and authoritative appearance to instill confidence in developers.*

Rationale: A professional appearance helps build trust and encourages developers to use the tool confidently for energy-efficient refactoring.
Fit Criterion: After their first encounter with the tool, the amount of representative developers that feel that it is a trustworthy and reliable solution for energy-efficient refactoring should be above the MIN_USER_CONFIDENCE.
Priority: High

LFR-ST 2. *The IDE plugin interface shall promote a calm and focused atmosphere, enhancing the developer's ability to concentrate on code improvements.*

Rationale: A calm environment reduces distractions and improves productivity, allowing developers to focus on their work effectively.
Fit Criterion: Developers should report feeling less distracted and more productive while using the tool, with the amount of developers indicating a positive change in their coding environment meeting the MIN_USER_CONFIDENCE.
Priority: Medium

LFR-ST 3. *The tool design shall be visually appealing and modern, aligning with contemporary software development tools.*

Rationale: A modern design improves user experience and satisfaction, making the tool more enjoyable to use.
Fit Criterion: The number of users that express satisfaction with the tool's visual design and layout after their initial interaction should meet the MIN_USER_CONFIDENC
Priority: Medium

11 Usability and Humanity Requirements

11.1 Ease of Use Requirements

UHR-EOU 1. *The tool shall have an intuitive user interface that simplifies navigation and functionality.*

Rationale: A simple, intuitive interface allows users to access the tool's key features quickly, improving usability and reducing the learning curve.
Fit Criterion: Users should be able to complete key tasks (e.g., parsing code, configuring settings) MAX_TASK_CLICKS limit.
Priority: High

UHR-EOU 2. *The tool shall provide clear and concise prompts for user input.*

Rationale: Clear instructions help users understand what inputs are required, minimizing confusion and errors during the process.
Fit Criterion: The amount of test users that report that prompts are straightforward and guide them effectively through the process should meet the MIN_USER_EOU.
Priority: High

11.2 Personalization and Internationalization Requirements

UHR-PSI 1. *The tool shall allow users to enable or disable detection of individual code smells.*

Rationale: Developers should be able to focus on relevant code smells for their specific project needs.
Fit Criterion: Users can successfully toggle detection of any smell type on or

off.
Priority: High

UHR-PSI 2. *The tool shall allow users to customize highlight colors for each detected code smell type.*

Rationale: Custom colors improve readability and accommodate different visual preferences.
Fit Criterion: Users can successfully change highlight colors for any smell type.
Priority: Medium

11.3 Learning Requirements

UHR-LRN 1. *The tool shall provide context-sensitive help that offers assistance based on the current user actions.*

Rationale: Context-sensitive help ensures that users can receive timely and relevant assistance, reducing confusion and improving usability.
Fit Criterion: Help resources should be accessible within MAX_TASK_CLICKS limit.
Priority: High

UHR-LRN 2. *The tool shall have an available YouTube video demonstrating installation.*

Rationale: Video tutorials provide visual learning resources that can make the installation process more accessible to users.
Fit Criterion: A YouTube video demonstrating installation should be present and easily accessible.
Priority: Low

11.4 Understandability and Politeness Requirements

UHR-UPL 1. *The tool shall communicate errors and issues politely and constructively.*

Rationale: Polite and constructive error messages reduce frustration and enhance the user experience, making the tool more approachable.
Fit Criterion: User feedback should reflect that at least the MIN_USER_EOU of users perceive error messages as helpful and courteous, rather than frustrating or vague.
Priority: Medium

11.5 Accessibility Requirements

UHR-ACS 1. *The tool shall provide high-contrast colour themes to improve visibility for users with visual impairments.*

Rationale: High-contrast themes ensure that visually impaired users can easily navigate and use the tool, enhancing accessibility.

Fit Criterion: Users should have access to at least 1 high contrast theme.
Priority: Low

12 Performance Requirements

12.1 Speed and Latency Requirements

PR-SL 1. *The tool shall analyze and detect code smells in the input code within a reasonable time frame.*

Rationale: Fast analysis ensures that developers do not experience significant delays while reviewing code.
Fit Criterion: The tool should complete the analysis for files up to 1,000 lines of code in under SMALL_FILE_TIME, and for files up to 10,000 lines in under LARGE_FILE_TIME.
Priority: High

PR-SL 2. *The refactoring process shall be executed efficiently without noticeable delays.*

Rationale: Fast refactoring ensures a smooth workflow for developers, preventing frustration during development.
Fit Criterion: The tool should refactor the code and generate output in under REFACTOR_TIME for small to medium-sized files (up to 5,000 lines).
Priority: Medium

12.2 Safety-Critical Requirements

PR-SCR 1. *The tool shall ensure that no runtime errors are introduced in the refactored code that could result in data loss or system failures.*

Rationale: Preventing runtime errors ensures system stability and reliability after refactoring.
Fit Criterion: The refactored code must produce valid Python code verified by the user upon accepting changes **Priority:** High

12.3 Precision or Accuracy Requirements

PR-PAR 1. *The tool shall maintain the functionality of the original provided code in all its recommended refactorings.*

Rationale: Ensuring functionality preservation is critical for refactorings to be reliable.
Fit Criterion: The tool should pass all tests from the user-provided test suite after refactoring, confirming that the original functionality remains intact.
Priority: High

PR-PAR 2. *The tool shall reliably identify code smells with minimal false positives and negatives.*

Rationale: High detection accuracy ensures that developers are not misled by incorrect or missed suggestions.
Fit Criterion: Detection accuracy should exceed DETECTION_ACC when validated against a set of known cases.
Priority: Medium

PR-PAR 3. *The tool shall produce valid refactored Python code as output or indicate that no possible refactorings were found.*

Rationale: Ensuring that the tool produces valid output is essential for maintaining code quality.
Fit Criterion: The output code is syntactically correct and adheres to Python standards, validated by an automatic linter.
Priority: Medium

12.4 Robustness or Fault-Tolerance Requirements

PR-RFT 1. *The tool shall gracefully handle unexpected inputs, such as invalid code or non-Python files.*

Rationale: Ensuring stability with error handling prevents tool crashes and improves user experience.
Fit Criterion: The tool should provide clear error messages and recover from input errors without crashing, ensuring stability.
Priority: High

PR-RFT 2. *The tool shall have fallback options if a specific refactoring attempt fails.*

Rationale: Providing fallback options ensures the tool remains functional even when a refactoring fails, reducing disruptions in development.
Fit Criterion: In the event of a failed refactoring, the tool should log the error and propose alternative refactorings without stopping the process.
Priority: Medium

12.5 Capacity Requirements

PR-CR 1. *The tool shall efficiently manage large codebases.*

Rationale: Efficient handling of large projects ensures that the tool remains usable for teams working with extensive codebases.
Fit Criterion: The tool must process projects with up to 100,000 lines of code within LARGE_CODE_BASE_TIME, maintaining performance standards.
Priority: High

12.6 Scalability or Extensibility Requirements

PR-SER 1. *The tool shall be designed to allow easy addition of new code smells and refactoring methods in future updates.*

Rationale: Extensibility ensures that the tool remains relevant and adaptable to future developments in coding standards and practices.
Fit Criterion: New code smells or refactorings can be incorporated with minimal changes to existing code, ensuring that current functionality remains intact.
Priority: Medium

12.7 Longevity Requirements

PR-LR 1. *The tool shall be maintainable and adaptable to future versions of Python and changing coding standards.*

Rationale: Ensuring the tool can be updated easily guarantees that it will remain useful and relevant over time.
Fit Criterion: The codebase should be well-documented and modular, facilitating updates with minimal effort.
Priority: Medium

13 Operational and Environmental Requirements

The Operational and Environmental Requirements define the conditions under which the system must function effectively. These requirements ensure that the system performs reliably within specified operational boundaries, such as compatibility, deployment, and environmental constraints. Additionally, this section addresses the external environmental factors that could influence the system. Meeting these requirements is critical to ensure the tool's proper operation and sustainability in various working environments.

13.1 Expected Physical Environment

OER-EP 1. *The product shall be used in temperatures ranging from* 10 °C - 35 °C.

Rationale: A computer's safe operating range is 10 °C - 35 °C (**?**). If the computer doesn't work then it is not possible to use the refactoring library.
Fit Criterion: The computer turns on, and no temperature warning is issued.
Priority: High

OER-EP 2. *The product shall be used in proximity to a stable power supply.*
Rationale: As a coding library, the product depends on the continuing operation of the computer system it is used on. Should the computer lose power, the refactoring library will see its processes halted.
Fit Criterion: The computer is connected to a power outlet or the computer possesses charge on its battery.
Priority: High

13.2 Wider Environment Requirements

OER-WE 1. *The system must align with widely used emissions standards (e.g., GRI 305, GHG, ISO 14064)* (**??**).

Rationale: Providing metrics tailored to these standards, makes the library reporting tool more attractive to users part of companies looking to reduce their ecological footprint.
Fit Criterion: The emissions tracked by the standards are present in the reported metrics.
Priority: Medium

13.3 Requirements for Interfacing with Adjacent Systems

OER-IAS 1. *The library should be compatible with the Visual Studio Code (VSCode) IDE.*

Rationale: Developers will be able to refactor code easily without leaving their working environment, therefore enhancing the accessibility and usability of the library.
Fit Criterion: An extension is available for installation in VSCode marketplace.
Priority: Medium

OER-IAS 2. *The library should support importing existing codebases and exporting refactored code and energy savings reports in standard formats (e.g., JSON, XML)*

Rationale: This ensures that users can easily integrate the library into their existing workflows without significant disruption.
Fit Criterion: Developers are able to refactor existing codebases and view relevant metrics.
Priority: Medium

13.4 Productization Requirements

OER-PR 1. *The library shall be package with PIP and made available to python users through the public package manager.*

Rationale: As a widely used package manager, PIP will be able to distribute the library to any users that wish to use it.
Fit Criterion: Users are able to install the library using `pip install`.
Priority: Medium

13.5 Release Requirements

OER-RL 1. *All core functionalities specified in the requirements must be implemented and tested, including energy consumption measurement, automated refactoring, and reporting features.*

Rationale: This will ensure that the library delivers the promised capabilities

to users.
Fit Criterion: Follows the steps outlined in the Verification and Validation (V&V) plan.
Priority: Medium

OER-RL 2. *The library must be ready for release by March 17th, 2025.*

Rationale: The library must be ready for final demonstration as a requirement of the McMaster University SFRWENG 4G06 Capstone course.
Fit Criterion: The project is ready for the final demonstration of the appointed date.
Priority: Low

14 Maintainability and Support Requirements

The following are defined as maintainability requirements:

14.1 Maintenance Requirements

MS-MNT 1. *The tool must allow new refactoring techniques to be added within one week of identification.*
Rationale: Rapid integration of new techniques ensures the tool remains up-to-date with evolving best practices in energy-efficient coding.
Fit Criteria: Developers can integrate new refactoring methods into the tool, and they are fully operational within NEW_REFACTOR_TIME.
Priority: Medium

MS-MNT 2. *The tool must be maintainable by developers who are not the original creators.*
Rationale: Ensuring that new developers can easily understand and modify the system reduces dependency on original developers and facilitates long-term maintenance.
Fit Criteria: Comprehensive documentation is available such as setup guides and code comments, allowing new developers to understand and modify the system within COMPREHENSION_TIME.
Priority: High

MS-MNT 3. *The tool must allow for easy rollback of updates in case of errors.*
Rationale: Quick rollback capabilities minimize downtime and user disruption in case an update introduces issues.
Fit Criteria: Any update can be reverted with minimal effort, ensuring the system returns to a stable state within ROLLBACK_TIME.
Priority: Medium

MS-MNT 4. *The tool must provide automated testing for all refactoring functions.*
Rationale: Automated testing ensures that changes do not introduce new bugs, maintaining the reliability and stability of the tool.

Fit Criteria: All refactoring methods have associated unit tests that run automatically with each code change, ensuring code coverage meets MIN_CODE_COVERAGE.
Priority: High

MS-MNT 5. *Each version of the library must maintain compatibility with the current releases of external libraries during its development phase.*
Rationale: Keeping external libraries up-to-date ensures compatibility and leverages improvements or security patches provided by library maintainers.
Fit Criteria: The system successfully integrates updates from external libraries without breaking existing functionality.
Priority: Medium

14.2 Supportability Requirements

This section is not needed for this project.

14.3 Adaptability Requirements

MS-AD 1. *The system must be compatible with the latest stable version of python (v3.13).*
Rationale: This allows the library to use the most up-to-date features, performance improvements, and security patches. Furthermore, it will ensure that users can integrate the library into modern projects without compatibility issues, reduces the risk of vulnerabilities.
Fit Criteria: The system passes all test cases when run in python 3.13.
Priority: High

MS-AD 2. *The system should offer backward compatibility with Python 3.10 and higher.*
Rationale: Offering backwards compatibility makes the library more accessible to users who may need to use the library in legacy codebases that can't easily be upgraded. Based on Python's versioning schedule, version 3.10 will not be considered end-of-life until nearly 2 years after the scheduled end of the development of this system.
Fit Criteria: All features of the system must work on Python 3.10 and higher versions, with minimal degradation in performance or functionality
Priority: Low

MS-AD 3. *The system must be able to detect and handle Python version-specific features and syntax.*
Rationale: Depending on the Python version of the source code, the refactorings should be made accordingly to not introduce compatibility errors.
Fit Criteria: The input source code should pass the given test cases.
Priority: High

MS-AD 4. *The energy measurement module must be adaptable to cloud-based systems.*
Rationale: Many modern applications are hosted and run in cloud environments so, it is essential for the energy measurement module to operate effectively in

cloud-based systems. This ensures that developers can accurately measure the energy consumption of their cloud-hosted applications.
Fit Criteria: The energy consumption module returns usable statistics related to the energy consumption of the cloud-application.
Priority: Medium

MS-AD 5. *The system must be deployable in both cloud-based environments and on-premise infrastructures to meet different user needs.*
Rationale: Whether the system is being used locally or in the cloud, developers still need access to the features offered by the system. Some developers might not even host their applications, and, even if they do, they might want to conduct local testing.
Fit Criteria: The system should be fully functional in both cloud and local environments, with minimal configuration changes.
Priority: Medium

MS-AD 6. *The system must support major operating systems, including Windows, macOS, and Linux.*
Rationale: Developers may not all use the same operating system (OS), it's actually unlikely (even impossible) that they do. Allowing users to use the library on their preferred OS will only increase the adoption of the system.
Fit Criteria: The system must pass all tests on Windows 10+, macOS 13+ (Ventura and later), and Linux distributions such as Ubuntu 22.04.5+ without requiring different codebases or significant modifications.
Priority: Low

MS-AD 7. *The system must provide seamless functionality across different versions of the same operating system.*
Rationale: Users do not adopt the latest version of an OS at the same rate and some programs are to complex or fragile to upgrade to newer systems.
Fit Criteria: The system runs without errors or crashes across multiple versions of Windows, macOS, and Linux, with no compatibility issues in handling file systems, dependencies, or libraries.
Priority: Low

MS-AD 8. *The system must offer consistent performance (e.g., refactoring speed, energy consumption measurements) regardless of the underlying operating system.*
Rationale: Having the system be OS dependent would invalidate the point of being compatible with multiple OSes and conflict with requirement MS-AD 6.
Fit Criteria: Performance metrics, such as time taken for refactoring and energy measurements, must not vary by more than OS_PERF_DIFF_LIMIT across different operating systems during testing.
Priority: Low

15 Security Requirements

15.1 Access Requirements

SR-AR 1. *The tool must be capable of protecting data during active sessions without requiring authentication of the user.*

Rationale: Ensuring secure handling of data during processing eliminates the need for user authentication, thereby improving the tool's accessibility. Organizational code is already access-controlled for developers, therefore authentication is unnecessary for the tool.
Fit Criterion: The tool allows users to submit code and view refactoring results without requiring login credentials, and no sensitive data is stored beyond the active session.
Priority: High

SR-AR 2. *Only the refactoring tool can communicate with the energy consumption tool.*
Rationale: Energy consumption tool is an internal abstracted service that is not directly needed by the user.
Fit Criterion: The refactoring tool does not include any exposed API endpoints to the energy consumption tool.
Priority: High

15.2 Integrity Requirements

SR-IR 1. *The tool must prevent unauthorized, external changes to the refactored code and energy reports.*

Rationale: The system must maintain code consistency and correctness of the original input and energy improvement data. Any corruption of the code could undermine trust in the tool.
Fit Criterion: The system must reject malformed data inputs, ensure data integrity during processing and transmission, and maintain comprehensive logging of system activities. It should safeguard against data corruption to maintain trust in the tool's outputs..
Priority: High

15.3 Privacy Requirements

SR-PR 1. *The tool must ensure that no personal user data, such as GitHub profile, commit history, working hours, or any identifiable information, is collected or stored during its operation.*

Rationale: Personal information, including coding habits, working patterns, and associated data (such as GitHub profile or commits), is sensitive and must not be used to ensure that users maintain full control over their personal data.

Fit Criterion: The tool analyzes code and provides refactoring suggestions without collecting, storing or transmitting any personal information about the user.
Priority: High

SR-PR 2. *Any data related to user submissions, energy reports and refactored results must be treated as confidential and handled securely throughout processing and transmission.*
Rationale: The tool must ensure user trust by safeguarding all user submissions and related data against unauthorized access or tampering. Secure handling of data during active sessions maintains the integrity and reliability of the tool.
Fit Criterion: The system processes user submissions, energy reports, and refactored results securely within the active session. No data is stored or retained after processing, ensuring all user information remains private and protected.
Priority: High

15.4 Audit Requirements

SR-AUR 1. *The tool should maintain a log of operational events related to the refactoring process, including code detetction, energy analysis, and refactoring.*

Rationale: The tool must ensure transparency in its operation by providing a record of key system actions, enabling the user to debug or review tool behavior.
Fit Criterion: The system generates a log of important operational events, such as the start and completion of analysis, energy measurements, and refactoring.

Priority: Medium

SR-AUR 2. *The tool should log user-triggered actions, such as code submissions and the initiation of refactoring processes, without storing any user-specific or identifiable data.*
Rationale: The tool must provide accountability for user interactions with the system, allowing the user to trace and verify the actions they have taken within the session.
Fit Criterion: The system logs events such as user code submissions, requests for refactoring, and access to refactoring results. These logs do not include personal identifiable user data.
Priority: Medium

15.5 Immunity Requirements

SR-AUR 1. *The tool must minimize exposure to external threats, ensuring that the refactoring process is protected from unauthorized interference.*

Rationale: Reducing reliance on external systems decreases vulnerability to external threats, such as malware or unauthorized access, and ensures a secure

and reliable refactoring process.
Fit Criterion: The system operates without any external server communication so as exposure to external threats is limited. This setup inherently reduces exposure to external security risks.
Priority: High

16 Cultural Requirements

The cultural requirements of this project include the following:

16.1 Cultural Requirements

CULT 1. *The tool must avoid using colours or symbols that could be culturally sensitive or offensive.*
Rationale: Ensuring cultural sensitivity in design helps avoid alienating or offending users from diverse backgrounds, which is critical for global acceptance and usability.
Fit Criteria: Conduct a cultural review to ensure that all icons and colours used in the tool are neutral and universally acceptable. **Priority:** Low

CULT 2. *The tool must not include content that could be considered culturally insensitive.*
Rationale: Avoiding culturally insensitive content ensures that the tool is respectful and inclusive, fostering a positive user experience across different cultures.
Fit Criteria: A cultural sensitivity review is conducted to ensure all content is appropriate for a global audience. **Priority:** Medium

17 Compliance Requirements

17.1 Legal Requirements

CR-LR 1. *The system must respect user privacy by avoiding the collection of any personal or identifiable data.*

Rationale: Ensuring privacy builds user trust and minimizes risks associated with handling sensitive information, even when specific privacy laws are not targeted.
Fit Criterion: The system avoids collecting or storing personal user data, including information about user activities and profiles, and operates entirely within the user's local environment.
Priority: Medium

17.2 Standards Compliance Requirements

CR-SCR 1. *The system must conform to established Python coding standards and best practices for maintainability, readability and quality.*

Rationale: Adhering to standards such as PEP 8 ensures the system's codebase remains consistent, maintainable and compatible with coding standards, facilitating future development and collaboration.
Fit Criterion: The system implements static code analysis and adheres to Python coding standards, including PEP 8.
Priority: Medium

18 Open Issues

This section outlines unresolved questions and challenges that may impact the development, functionality, or integration of the system. These issues require further research, discussion, or testing to ensure successful project completion.

- Further research is needed to determine the optimal balance between energy efficiency and code readability. While refactoring may improve energy consumption metrics, it could inadvertently make the codebase less maintainable and more difficult to expand in the long term.
- The same can be said when it comes to performance. More energy efficient code might actually end up being less efficient when it comes to time and space complexity.

19 Off-the-Shelf Solutions

19.1 Ready-Made Products

- **Pylint:** A widely used static code analysis tool that detects various code smells in Python. It can be integrated into the refactoring tool to help identify inefficiencies in the code.
- **Flake8:** Linter that combines checks for style guide enforcement and code quality. Flake8 can assist in maintaining code standards while the tool focuses on energy efficiency.
- **PyJoule:** A tool for measuring the energy consumption of Python code. This product can provide essential data to evaluate the impact of refactorings on energy usage.

19.2 Reusable Components

- **Rope:** A library for Python that provides automated refactoring capabilities, helping streamline the process of improving code quality.

19.3 Products That Can Be Copied

- **SonarQube:** An open-source platform designed for continuous inspection of code quality. It helps developers manage code quality and security by analyzing source code to identify potential issues. Its architecture and methods for detecting code smells could be adapted to focus specifically on energy efficiency.

20 New Problems

20.1 Effects on the Current Environment

The introduction of the energy efficiency refactoring tool may lead to several changes in the current development environment. These effects include:

1. The tool temporarily increases CPU and memory usage while running. The tool aims to optimize energy efficiency in code however it takes energy to run - in large codebases this could be significant energy and impact the performance of other applications running concurrently.

2. The tool may have its own dependencies that now need to be included in the app or installed into the current system. Think Pysmells, Pyjoule etc.

20.2 Effects on the Installed Systems

1. Existing systems may need to be evaluated for compatibility with the new tool. Older versions of Python or other needed dependencies may not support the tool.

2. The refactoring process could lead to variations in the performance of existing applications

3. As the tool updates existing code, thorough testing will be needed to ensure everything still works correctly. This may require more effort from QA teams and additional time and resources to check the updated code.

20.3 Potential User Problems

1. Users may face difficulties in understanding how to effectively utilize the tool, particularly if they are not familiar with concepts like code smells and refactoring techniques. This learning curve may lead to initial frustration or reduced productivity.

2. Some users may be resistant to adopting new tools or processes, particularly if they perceive the existing workflows as sufficient. This resistance could hinder the tool's successful implementation and limit its overall effectiveness.

3. Users may misinterpret the output reports generated by the tool, such as energy savings or performance metrics. If users do not fully understand how to interpret these results, it could lead to incorrect conclusions about the tool's impact on their code.

4. There is a risk that users might become overly reliant on the tool for refactoring without fully understanding the underlying principles. This could result in poor coding practices if users do not engage in thoughtful analysis of the suggested changes.

20.4 Limitations in the Anticipated Implementation Environment That May Inhibit the New Product

1. **Limited Computational Resources:** Environments with restricted computational power may face challenges when running the tool, especially for large codebases. Limited resources could result in longer processing times or failures during analysis and refactoring.
2. **Lack of Test Coverage:** If existing codebases lack comprehensive test suites, validating the functionality of refactored code may become challenging. Without adequate tests, it will be difficult to ensure that the tool's changes do not introduce new issues.

20.5 Follow-Up Problems

1. **Ongoing Maintenance:** The tool will need regular updates to stay compatible with new programming languages or standards, adding to the workload.
2. **Performance Trade-offs:** Users may find that while some refactorings improve energy efficiency, they could negatively impact other performance metrics, such as execution speed.

21 Tasks

21.1 Project Planning

- **Development Approach** The team will use an agile development approach with the following high-level process:
  1. Initial requirements gathering and product backlog creation
  2. Sprint planning and execution
  3. Regular testing and quality assurance
  4. Stakeholder reviews and feedback
  5. Iterative refinement
  6. Release planning and deployment
- **Key Tasks**
  - Form cross-functional development team (already completed)

  - Create initial product backlog and prioritize features
  - Set up development environments and tools
  - Establish CI/CD pipeline using GitHub Actions
  - Develop core functionality:
    * Determine code smells to address for energy-saving
    * Implement code smell detection
    * Develop appropriate refactorings for detected smells
    * Measure energy consumption before and after refactoring
    * Ensure original code functionality is preserved
  - Build out additional features iteratively
  - Conduct regular testing (unit, integration, user acceptance)
  - Refine based on stakeholder feedback
  - Present final solution to stakeholders

- **Timeline Estimate**
  - Requirements Document (Revision 0): October 9th, 2024
  - Hazard Analysis (Revision 0): October 23rd, 2024
  - Verification & Validation Plan (Revision 0): November 1st, 2024
  - Proof of Concept: November 11th-22nd, 2024
  - Design Document (Revision 0): January 15th, 2025
  - Project Demo (Revision 0): February 3rd-14th, 2025
  - Final Demonstration: March 17th-30th, 2025
  - Final Documentation: April 2nd, 2025
  - Capstone EXPO: TBD
- **Resource Estimates** The team consists of 5 members who will all function as developers, sharing responsibilities for creating issues, coding, testing, and documentation.
- **Key Consideration**
  - Data migration may be necessary for existing systems

    - A phased development approach will help minimize major setbacks
    - Regular stakeholder involvement will ensure alignment with business needs

- **Documentation Process**
    - Pull changes from `docs` (epic documentation branch)
    - Create a working branch with format [main contributor name]/[descriptive topic]
    - Commit changes with descriptive names
    - Create unit tests for changes
    - Create a pull request to merge changes into an epic branch
    - Wait for all tests run with GitHub Actions to pass
    - Wait for at least two approvals from teammates
    - Merge changes into the target branch

By following this agile approach and development process, the team aims to deliver a high-quality product iteratively while maintaining flexibility to adapt to changing requirements

21.2 Planning of the Development Phases

The planning of the development phases is based on the deliverables submissions as follows:

1. **Requirements Phase**
    - Deliverable: Requirements Document (Revision 0)
    - Due Date: October 9th, 2024
2. **Risk Assessment Phase**
    - Deliverable: Hazard Analysis (Revision 0)
    - Due Date: October 23rd, 2024
3. **Verification and Validation Planning**
    - Deliverable: Verification & Validation Plan (Revision 0)
    - Due Date: November 1st, 2024
4. **Proof of Concept Implementation**

- Period: November 11th-22nd, 2024

5. **Design Phase**
   - Deliverable: Design Document (Revision 0)
   - Due Date: January 15th, 2025
6. **Initial Implementation and Demo**
   - Deliverable: Project Demo (Revision 0)
   - Period: February 3rd-14th, 2025
7. **Final Implementation and Testing**
   - Deliverable: Final Demonstration
   - Period: March 17th-30th, 2025
8. **Project Closure**
   - Deliverable: Final Documentation
   - Due Date: April 2nd, 2025
9. **Project Presentation**
   - Event: Capstone EXPO
   - Date: TBD

22 Migration to the New Product

22.1 Requirements for Migration to the New Product

The migration to the new tool will involve minimal disruption to existing workflows, as the product is designed to integrate seamlessly with Visual Studio Code. Developers will be able to continue using their current workflows, with the addition of the refactoring tool as a library that can be installed directly via Python's package manager (pip) and an IDE plugin for an enhanced development experience. The minimal installation and integration requirements ensure ease of adoption for developers without needing to overhaul existing infrastructure or processes.

- **Installation Phase:** Developers can install the refactoring library directly using pip: `pip install ecooptimizer` The IDE plugin can be installed by users through the Visual Studio Code marketplace.

- **Parallel Running:** There is no need for a phased implementation or parallel running of old and new systems. The tool can be directly integrated without affecting existing code or infrastructure.
- **Manual Backups:** Since the tool does not alter data but rather provides refactoring suggestions, no manual backups are required before installation.
- **Phased Rollout:** If required, developers can start using the refactoring library first, followed by the IDE plugin.

22.2 Data That Has to be Modified or Translated for the New System

Not Applicable

23 Costs

The total cost of developing this project is primarily based on the effort involved by the development team, given that the tools and platforms used (GitHub, open-source libraries) are free. The project must be completed within the academic year (MVP ready by February 2025), which warrants smart planning and efficient resource allocation.

23.1 Metrics for Estimation

To estimate the total cost in terms of time and effort, the following key metrics have been considered:

- Number of input/output flows for the system.
- Number of business events.
- Number of product use cases.
- Number of functional and non-functional requirements.
- Number of constraint requirements.

23.2 Estimation Approach

Each deliverable has been assessed to estimate the time it will take to implement based on the development environment. Early cost estimates are based on general knowledge of the system and refined as the team gradually gets a better understanding of the scope.

23.3 Cost Breakdown

- **Development Effort:** Based on the team size and project timeline, following time allocation is estimated:
  - Initial Research & Setup: 50 hours per team member.
  - Core development (e.g., refactoring tool, plugin): 300 hours per team member.
  - Testing and debugging: 80 hours per team member.
  - Documentation & finalization: 100 hours per team member.
  - **Total estimated effort:** 530 hours per team member.
- **Tools and Software:**
  - GitHub (for CI/CD): Free account, with no cost expected.
  - Open-source libraries: No associated costs.
- **Testing Environment:**
  - The testing of the refactoring tool and energy consumption measurements is planned to be carried out using free tools on open source projects, with no additional cost expected.

23.4 Estimated Cost

The above discussion indicates that the total effort will be approximately 2650 team hours spread across the project phases. These estimates could be refined using more detailed information as the project progresses.

24 User Documentation and Training

24.1 User Documentation Requirements

1. **User Manual**
   - **Purpose:** Provide comprehensive guidance on how to use the refactoring library
   - **Target Audience:** Software developers integrating the library into their projects
   - **Content:** Installation instructions, API reference, usage examples, and best practices
2. **Technical Specification**

- **Purpose:** Detail the library's architecture, algorithms, and integration points
- **Target Audience:** Technical leads and architects evaluating the library
- **Content:** System design, performance characteristics, and technical limitations

3. **Quick Start Guide**

   - **Purpose:** Enable rapid adoption and basic usage of the library
   - **Target Audience:** New users looking to quickly implement the library
   - **Content:** Concise setup instructions and simple usage examples

24.2 Training Requirements

For end-users, formal training is not required. The tool is designed to be simple and intuitive, with the assumption that primary users with experience in Python and software development will have the necessary technical expertise to use the library effectively. The refactoring library operates through clear, well-documented interfaces and requires minimal setup, allowing users to quickly incorporate it into their workflow without additional training.

Furthermore, the tool will come with comprehensive user documentation that includes step-by-step instructions on how to set up, integrate, and use the library within their existing development processes. This documentation will provide examples of common use cases and a clear explanation of the features of the tool, ensuring that users can get started easily and efficiently.

25 Waiting Room

This section lists potential features and enhancements that are not critical for the initial release of the system but may be considered for future updates. These ideas represent additional functionalities that could improve the tool's performance, user experience, or integration with other systems.

WTRM 1. *The IDE plugin must provide interactive tips and progress indicators to guide users during the refactoring process.*

Rationale: This helps prevent user confusion during the refactoring process and enhances the user experience by providing clear, real-time feedback.
Fit Criterion: Tips and progress indicators display automatically when users initiate a refactoring action in the VSCode interface.
Priority: Medium

WTRM 2. *The tool must provide automated error reporting with user consent.*

Rationale: Automated error reporting helps developers quickly identify and

address issues, improving the tool's reliability and reducing downtime with an option to include additional comments for context.
Fit Criterion: Users are prompted to send error reports when an issue occurs,
Priority: Medium

WTRM 3. *The tool must allow users to define custom refactoring rules or preferences within the plugin.*

Rationale: This would allow developers more flexibility and control, enabling them to apply refactorings that best suit their projects while maintaining energy efficiency.
Fit Criterion: Users can create a configuration file that describes their refactoring preferences.
Priority: High

WTRM 4. *The tool should provide a dashboard that offers deeper insights into refactoring decisions, showing side-by-side comparisons of energy consumption before and after refactoring, along with performance metrics.*

Rationale: This would help users better understand the impact of refactoring on both energy consumption and performance.
Fit Criterion: Users are able to access energy metrics for all their projects on a centralized platform.
Priority: High

WTRM 5. *The tool show provide a plugin that integrates with multiple popular IDEs and syncs information across them through a centralized database.*

Rationale: Supporting multiple IDEs allows developers to use the tool within their preferred coding environments, enhancing user experience and adoption.
Fit Criterion: Integration guides are provided for each supported IDE, with successful installation and functionality confirmed through user testing.
Priority: Low

WTRM 6. *The refactoring library must be able to provide energy efficient refactorings for a wide variety of program languages (e.g., Java, C/C++, C#, JavaScript, TypeScript, Go, Rust, etc.)*

Rationale: Users around the world code projects in multitudes of languages depending on the context of their work. Being able to refactor in most popular languages would allow for the widespread usage of the tool.
Fit Criterion: The tool accepts and tailors code from the detected/specified language.
Priority: Low

WTRM 7. *A reinforcement learning model must be developed to learn from the most energy-efficient refactorings and user preferences over time.*

Rationale: The model must evolve to improve its refactoring suggestions by

identifying patterns in energy-efficient code transformations and user preferences. By adapting its recommendations, the system will refine its ability to balance energy savings with maintainability and user preferences.
Fit Criterion: The reinforcement learning model is trained using gathered refactoring data and energy consumption measurement. Over multiple iterations, the model should improve its ability to predict refactorings that maximize energy efficiency while aligning with user choices.
Priority: Medium

WTRM 8. *The reinforcement learning model for the tool should accept human feedback in the reinforcement learning process.*

Rationale: This will allow users to guide the system's refactoring decisions based on developer expertise and preferences. Moreover, it will balance automated refactoring with human oversight to ensure that complex refactoring decisions align with the project's goals and constraints.
Fit Criterion: The language model accepts human feedback and incorporates it into its future decisions.
Priority: Medium

26 Ideas for Solution

In this section, we capture various ideas for implementing the project requirements. These ideas provide technical approaches and methods for achieving the energy-efficient refactoring of Python code through the refactoring tool and IDE integration. By capturing these ideas now, we ensure that potential technical approaches are not lost and can be referred to during the design and development phase.

- **Refactoring Library Implementation via AST Parsing:** Use Python's built-in `ast` (Abstract Syntax Tree) module to analyze code structure. The refactoring library can traverse the AST to detect inefficiencies and apply refactorings without modifying code functionality.

- **IDE Plugin for Refactoring – Leveraging VS Code API:** Build an extension for Visual Studio Code that hooks into the editor's event system to provide real-time refactoring suggestions.

Appendix — Reflection

1. *What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project?*

    - Learn how reinforcement learning works as well as how to incorporate it into our project.
    - Understanding of Python's performance characteristics and common code smells
    - Experience in using libraries like rope for automated refactoring and familiarity with integrating linters such as Pylint or Flake8 into the development workflow.
    - Ability to develop algorithms that analyze and compare different refactoring strategies, using tools like PyJoule for energy profiling.
    - Proficiency in JavaScript or TypeScript, as most VS Code extensions are developed using these languages.

2. ***(team)*** *For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill?*

    - *Learn how reinforcement learning works as well as how to incorporate it into our project.*
        - Watch online courses on reinforcement learning.
        - Read articles/books teaching reinforcement learning.
        - Look through open source python projects that use reinforcement learning tools such as PyTorch or TensorFlow.
        - Read through the documentation for reinforcement learning tools.
    - *Understanding of Python's performance characteristics and common code smells.*
        - Enrol in courses like Effective Python or Python Performance Optimization on platforms like Coursera, edX, or Udemy.
        - Work on real-world projects or contribute to open-source Python projects. Regular code reviews with a focus on performance will help identify common code smells and inefficiencies.
    - *Experience in using libraries like rope for automated refactoring and familiarity with integrating linters such as Pylint or Flake8 into the development workflow.*
        - Set up a personal or team-based project that uses rope for refactoring and integrates linters (Pylint or Flake8). Experimenting with these tools in a real project will provide direct experience with their workflows and limitations
        - Read through documentation for the tools.

    - Follow comprehensive tutorials or attend workshops focused on Python tooling.

- *Ability to develop algorithms that analyze and compare different refactoring strategies, using tools like PyJoule for energy profiling.*
    - Conduct research on energy-efficient algorithms and their application in Python refactoring. Experiment with PyJoule to analyze the energy impact of different code structures.
    - Work under the guidance of a mentor or collaborate with experts in energy-efficient computing.

- *Proficiency in JavaScript or TypeScript, as most VS Code extensions are developed using these languages.*
    - Practise by building simple VS Code extensions using JavaScript or TypeScript using tutorials on YouTube.
    - Take online courses specifically tailored to JavaScript or TypeScript for extension development

2. ***(Individual)** Of the identified approaches, which will each team member pursue, and why did they make this choice?*

Ayushi Amin Reflection

Learn how reinforcement learning works as well as how to incorporate it into our project.

I would choose looking through open-source Python projects that use reinforcement learning tools like PyTorch or TensorFlow. It lets me see real-world applications, understand how others are using reinforcement learning, and gives me concrete examples to work with, which helps me learn more effectively through hands-on experience.

Understanding of Python's performance characteristics and common code smells.

I would go with practical experience, contributing to open-source projects, where I can directly apply performance optimizations and get feedback through code reviews. Real-world application will help cement these concepts better.

Experience in using libraries like rope for automated refactoring and familiarity with integrating linters such as Pylint or Flake8 into the development workflow.

I would go for hands-on implementation in my projects. I learn best by doing, so I would set up a project where I can integrate rope for refactoring and Pylint or Flake8 for linting. This would give me a chance to really see how these tools fit into the development process, and I could refine my setup as I work.

Ability to develop algorithms that analyze and compare different refactoring strategies, using tools like PyJoule for energy profiling.

I would prefer diving into the research around energy-efficient algorithms and then trying things out with PyJoule. I enjoy exploring new tools and methods, so experimenting with how different code changes affect energy use would be a good challenge for me.

Proficiency in JavaScript or TypeScript, as most VS Code extensions are developed using these languages.

I like learning by creating, so I would build an extension right away. This way, I would get immediate experience with JavaScript and TypeScript while learning the VS Code extension framework.

Mya Hussain Reflection

To advance my skills for the capstone project, I plan on deepening my understanding of Python's performance characteristics and common code smells while leveraging my existing knowledge of Python. I plan to read up on different potential code smells, detection methods and possible refactoring remedies and immediately apply these insights to help develop a proof of concept (POC) for our project. For mastering refactoring tools like Rope and integrating linters such as Pylint and Flake8, I will set up a local project environment to experiment with these tools specifically for our POC. Additionally, I'll explore energy-efficient algorithms and use PyJoule to analyze how different refactoring strategies impact performance in our specific context. To enhance my proficiency in JavaScript or TypeScript for developing VS Code extensions, I plan on watching online tutorials and perhaps take an online Javascript/Typescript course as it has been a while since I have been familiar with the syntax. I will also watch tutorials on how to set up a VSCode extension as this project will be my first time doing it. I'm very excited to start coding up a POC and move on from the documentation stage. With every requirement we write the team feels more and more pressure to make sure our idea will work. It's hard to write so much about a product that doesn't yet exist without having the proof that it can exist. Or at least my mind is having a hard time writing so much about what currently is an empty repository.

Tanveer Brar Reflection

Learn how reinforcement learning works as well as how to incorporate it into our project.

I plan to watch online courses on reinforcement learning. I prefer learning visually whenever possible. Since this is a fairly new concept for me, I think online courses would be the best, providing a mix of lectures and self-assignments. A lot of courses also provide access to an online learning community which is perfect for a beginner like me.

Understanding of Python's performance characteristics and common code smells.

For this, I will contribute to open source Python projects. Live projects are a good way to expose to practical coding challenges related to performance and code smells. It will also allow me to learn from more experience contributors and spot common code smells so that I can write scalable and efficient Python code.

Experience in using libraries like rope for automated refactoring and familiarity with integrating linters such as Pylint or Flake8 into the development workflow.

I will follow comprehensive tutorials to get experience in using Rope for automatic refactoring. Comprehension tutorials provide guided experience with Python tooling, helping to explore features of tools like Rope, Pylint, and Flake8 in-depth. The tutorials are structured, therefore helping to quickly gain familiarity with functionality and usage in real-world scenarios.

Ability to develop algorithms that analyze and compare different refactoring strategies, using tools like PyJoule for energy profiling.

For this I will conduct research on energy-efficient algorithms and their application in Python refactoring. This will give me the option to follow a hands-on approach which allows for comparing different strategies and asses their energy consumption in multiple scenarios.

Proficiency in JavaScript or TypeScript, as most VS Code extensions are developed using these languages.

For this I will practise building simple VS Code extensions using JavaScript or TypeScript through YouTube tutorials. It can help me gain hands-on experience with the languages in a real-world context. These practical exercises can be used to not only solidfiy my understanding of JavaScript/TypeScript but also familiarize me with how these languages are used to create functional VS Code extensions, enhancing both my coding skills and extension development proficiency.

Sevhena Walker Reflection

Learn how reinforcement learning works as well as how to incorporate it into our project.

For this topic, I plan on watching online courses and tutorials since it's a subject I have basically no experience in. Watching videos is the better option for me in this case since having someone explain the knowledge and providing examples will allow me to grasp the material faster.

Understanding of Python's performance characteristics and common code smells.

I believe reading articles and going through documentation will be best for me for this topic. I already have a good base in programming and Python specifically so written documents will allow me to digest the new information at my own pace.

Experience in using libraries like rope for automated refactoring and familiarity with integrating linters such as Pylint or Flake8 into the development workflow.

When it comes to libraries and modules, I tend to fare well by just reading the documentation and looking over code examples using those tools.

Ability to develop algorithms that analyze and compare different refactoring strategies, using tools like PyJoule for energy profiling.

For this topic, experimentation will be my best option as it is not really a matter of learning something completely new but applying a new tool in a known environment.

Proficiency in JavaScript or TypeScript, as most VS Code extensions are developed using these languages.

I already have some experience working with those languages, so I will only need to refer to the documentation for the occasional refresher on syntax and such.

Nivetha Reflection

Learn how reinforcement learning works as well as how to incorporate it into our project.

I prefer to look through open source Python projects that use reinforcement learning tools to get used to it. While documentation and tutorials are readily available online whenever I need to reference them, I'm interested in seeing how tools such as PyTorch are applied in a live project so that I can get a clear understanding of their use. If possible, I will also contribute to these open source projects to implement these tools in a side project before doing for the capstone. This way I can be ready to implement in our project.

Understanding of Python's performance characteristics and common code smells.

I will enrol in online courses since they include structured learning for these tools to optimize Python code. These courses also include hands on exercises that I can practice solidifying my skills for identifying performance bottlenecks and avoiding code smells.

Experience in using libraries like rope for automated refactoring and familiarity with integrating linters such as Pylint or Flake8 into the development workflow.

I plan to follow comprehensive tutorials as these can break down complex topics into manageable subtopics, therefore helping in gradually building experience in these libraries. I can follow them at my own pace, so I have time to both experiment and troubleshoot. This way I can fully grasp the process of integrating these into Python projects.

Ability to develop algorithms that analyze and compare different refactoring strategies, using tools like PyJoule for energy profiling.

I plan to conduct research on analyzing the energy impact on various code structures using PyJoules. This way I can gather data to make informed refactoring decisions. This also opens up the opportunity to develop more efficient algorithms that can optimize performance and energy usage in Python applications.

Proficiency in JavaScript or TypeScript, as most VS Code extensions are developed using these languages.

Taking online courses tailored to JavaScript or TypeScript for extension development would give me a deeper understanding of how these languages are used in developing VS Code extensions. These focused courses include key concepts and best practices, helping me quickly become comfortable in the exact skills needed for extension development.