Software Requirements Specification for Software Engineering: subtitle describing software

Team 6, EcoOptimizers

Nivetha Kuruparan
Sevhena Walker
Tanveer Brar
Mya Hussain
Ayushi Amin

October 6, 2024

Contents

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

1 Purpose of the Project

1.1 User Business

Insert your content here.

1.2 Goals of the Project

Insert your content here.

2 Stakeholders

2.1 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.

2.2 Customer

The customers of this product are all the **software developers** that use it to improve the energy efficiency of their codebase. They will be the primary users of the product and, therefore, will offer critical feedback on its effectiveness. Suggestions for improvement and/or additional features may come from this stakeholder.

2.3 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 their 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 optimised using the refactoring library. They might experience more responsive, efficient software, particularly in mobile or embedded environments where battery life is a key concern. 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.

2.4 Hands-On Users of the Project

Software Developers

- **User Role**: Integrate library into 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

2.5 Personas

Persona 1
Raven Reyes, a 34-year-old senior software developer, has over a decade of experience in backend development. Now working at a SaaS company focused on sustainability, Raven faces the challenge of reducing the energy consumption of their software. Manually refactoring code is time-consuming, especially when trying to pinpoint which parts of the code are draining the most energy. With tight deadlines and a need to balance sustainability with performance, Raven seeks a tool that automates energy-efficient refactoring and provides clear metrics.

Persona 2
Draco Malfoy, a 29-year-old sustainability analyst at a large tech company, is tasked with reducing the environmental impact of the company's operations. With expertise in corporate sustainability, Jordan's role is to track and report energy consumption and carbon emissions, but they face challenges quantifying the data coming from the software development team. With the company's push toward greener technology, Jordan needs clear, easy-to-understand metrics on how the software team's refactoring efforts are improving energy efficiency. Bridging the gap between the sustainability and development teams, Jordan relies on these insights to report progress on key sustainability goals to executives, ensuring that both technical improvements and environmental targets are aligned.

2.6 Priorities Assigned to Users

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

2.7 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.

2.8 Maintenance Users and Service Technicians

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

3 Mandated Constraints

3.1 Solution Constraints

Insert your content here.

3.2 Implementation Environment of the Current System

Insert your content here.

3.3 Partner or Collaborative Applications

Insert your content here.

3.4 Off-the-Shelf Software

Insert your content here.

3.5 Anticipated Workplace Environment

Insert your content here.

3.6 Schedule Constraints

Insert your content here.

3.7 Budget Constraints

Insert your content here.

3.8 Enterprise Constraints

Insert your content here.

4 Naming Conventions and Terminology

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

Insert your content here.

5 Relevant Facts And Assumptions

5.1 Relevant Facts

Insert your content here.

5.2 Business Rules

Insert your content here.

5.3 Assumptions

Insert your content here.

6 The Scope of the Work

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

6.3 Work Partitioning

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

6.4 Specifying a Business Use Case (BUC)

6.4.1 Business Use Case Scenario 1

Event Name: 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 original Python code submitted
3. Tool analyzes the code for inefficiencies (PySmells)
4. Python code is provided to 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 original code
7. Refactored code is received by the user

Sub Variation:

- *4a:* If no PySmells 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

6.4.2 Business Use Case Scenario 2

Event Name: 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

6.4.3 Business Use Case Scenario 3

Event Name: 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 given code back to the refactoring tool

6.4.4 Business Use Case Scenario 4

Event Name: Testing and Validation of Refactored Code
Input: Refactored Code
Output: Test Results
Pre-condition: Energy consumed for refactored code is less than 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 user

Sub Variation:

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

6.4.5 Business Use Case Scenario 5

Event Name: 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

6.4.6 Business Use Case Scenario 6

Event Name: 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

7.1 Business Data Model

Insert your content here.

7.2 Data Dictionary

Insert your content here.

8 The Scope of the Product

8.1 Product Boundary

Insert your content here.

8.2 Product Use Case Table

Insert your content here.

8.3 Individual Product Use Cases (PUC's)

Insert your content here.

9 Functional Requirements

9.1 Functional Requirements

1. **Requirement:** The tool must accept Python source code files.

   Fit Criteria: The tool successfully processes valid Python files without errors and provides feedback for invalid files.

2. **Requirement:** The tool must identify specific code smells that can be targeted for energy saving.

   Fit Criteria: The tool should be able to detect and report at least 80% of the follwing code smells using predefined rules or existing linters. Code smells include: Large Class (LC), Long Parameter List (LPL), Long Method (LM), Long Message Chain (LMC), Long Scope Chaining (LSC), Long Base Class List (LBCL), Useless Exception Handling (UEH), Long Lambda Function (LLF), Complex List Comprehension (CLC), Long Element Chain (LEC), Long Ternary Conditional Expression (LTCE).

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

Fit Criteria: The suggested refactored code demonstrates a measurable improvement in energy measured in joules.

4. **Requirement:** The tool must implement an algorithm to choose the most optimal refactoring based on measured energy consumption.

   Fit Criteria: The algorithm evaluates multiple refactoring options and selects the one that results in the lowest energy consumption for the given code smell.

5. **Requirement:** The tool must produce valid refactored python code as output or indicate that no possible refactorings were found.

   Fit Criteria: The output code is syntactically correct and adheres to Python standards, validated by an automatic linter.

6. **Requirement:** The tool must report to the user any discrepancies between the original and suggested refactored code.

   Fit Criteria: Discrepancy reports to user clearly highlight differences in outputs

7. **Requirement:** The tool must allow users to input their original test suite as a required argument.

   Fit Criteria: Users can specify a path to their test suite, and the tool recognizes and utilizes it for testing the refactored code.

8. **Requirement:** The tool must ensure that the original functionality of the code is preserved after refactoring.

   Fit Criteria: The tool runs the original test suite against the refactored code, and passes 100% of the tests.

9. **Requirement:** The tool must be compatible with various Python versions and common libraries.

   Fit Criteria: The tool operates correctly with the latest two major versions of Python (e.g., Python 3.8 and 3.9) and commonly used libraries.

10. **Requirement:** The tool must generate comprehensive reports on detected smells, refactorings applied, energy consumption measurements, and testing results.

    Fit Criteria: Reports are clear, well-structured, and provide actionable insights, with users able to easily understand the results.

11. **Requirement:** The tool must provide comprehensive documentation and help resources.

    Fit Criteria: Documentation covers installation, usage, and troubleshooting, receiving positive feedback for clarity and completeness from users.

10 Look and Feel Requirements

10.1 Appearance Requirements

Insert your content here.

10.2 Style Requirements

Insert your content here.

11 Usability and Humanity Requirements

11.1 Ease of Use Requirements

1. **Requirement:** The tool must have an intuitive user interface that simplifies navigation and functionality.

   Fit Criteria: Users should be able to complete key tasks (e.g., parsing code, configuring settings) within three clicks or less.

2. **Requirement:** The tool should provide clear and concise prompts for user input.

   Fit Criteria: At least 90% of test users should report that prompts are straightforward and guide them effectively through the process.

11.2 Personalization and Internationalization Requirements

1. **Requirement:**The tool should allow users to customize settings to match their preferences (e.g., refactoring styles, detection sensitivity).

   Fit Criteria: Users should be able to save and load custom configurations easily.

2. **Requirement:** User guide must be available in French and English.

   Fit Criteria: French and english installation and use instructions available.

11.3 Learning Requirements

1. **Requirement:** The tool must have an availible youtube video demonstrating installation.

   Fit Criteria: Youtube video present and easily accessible

2. **Requirement:** The tool should provide context-sensitive help that offers assistance based on the current user actions.

   Fit Criteria: Help resources should be accessible within 1-3 clicks.

11.4 Understandability and Politeness Requirements

1. **Requirement:** The tool should communicate errors and issues politely and constructively.

   Fit Criteria: User feedback should reflect that at least 80% of users perceive error messages as helpful and courteous, rather than frustrating or vague.

2. **Requirement:** The tool should provide context-sensitive help that offers assistance based on the current user actions.

   Fit Criteria: Help resources should be accessible within 1-3 clicks.

11.5 Accessibility Requirements

1. **Requirement:** The tool should provide high-contrast color themes to improve visibility for users with visual impairments.

   Fit Criteria: Users should have access to at least 1 high contrast theme.

2. **Requirement:** The tool should offer audio cues for important actions and alerts to assist users with use and navigation.

   Fit Criteria: At least 70% of users should report that the audio cues enhance their understanding of important notifications or actions.

12 Performance Requirements

12.1 Speed and Latency Requirements

1. **Requirement:** The tool must analyze and detect code smells in the input code within a reasonable time frame.

   Fit Criteria: The tool should complete the analysis for files up to 1,000 lines of code in under 5 seconds, and for files up to 10,000 lines in under 30 seconds.

2. **Requirement:** The refactoring process must be executed efficiently without noticeable delays.

   Fit Criteria: The tool should refactor the code and generate output in under 10 seconds for small to medium-sized files (up to 5,000 lines).

12.2 Safety-Critical Requirements

1. **Requirement:** The tool must ensure that no runtime errors are introduced in the refactored code that could result in data loss or system failures.

Fit Criteria: The tool should pass all tests from the user-provided test suite after refactoring, confirming that the original functionality remains intact. The output code is syntactically correct and adheres to Python standards, validated by an automatic linter.

12.3 Precision or Accuracy Requirements

1. **Requirement:** The tool must reliably identify code smells with minimal false positives and negatives.

   Fit Criteria: Detection accuracy should exceed 90% when validated against a set of known cases.

2. **Requirement:** The tool must maintain the functionality of the original provided code in all its reccommended refactorings.

   Fit Criteria: The tool should pass all tests from the user-provided test suite after refactoring, confirming that the original functionality remains intact.

3. **Requirement:** The tool must produce valid refactored python code as output or indicate that no possible refactorings were found.

   Fit Criteria: The output code is syntactically correct and adheres to Python standards, validated by an automatic linter.

12.4 Robustness or Fault-Tolerance Requirements

1. **Requirement:** The tool should gracefully handle unexpected inputs, such as invalid code or non-Python files.

   Fit Criteria: The tool should provide clear error messages and recover from input errors without crashing, ensuring stability.

2. **Requirement:** The tool must have fallback options if a specific refactoring attempt fails.

   Fit Criteria: In the event of a failed refactoring, the tool should log the error and propose alternative refactorings without stopping the process.

12.5 Capacity Requirements

1. **Requirement:** The tool should efficiently manage large codebases.

   Fit Criteria: The tool must process projects with up to 100,000 lines of code within 2 minutes, maintaining performance standards.

12.6 Scalability or Extensibility Requirements

1. **Requirement:** The tool should be designed to allow easy addition of new code smells and refactoring methods in future updates.

   Fit Criteria: New code smells or refactorings can be incorporated with minimal changes to existing code, ensuring that current functionality remains intact.

12.7 Longevity Requirements

1. **Requirement:** The tool should be maintainable and adaptable to future versions of Python and changing coding standards.

   Fit Criteria: The codebase should be well-documented and modular, facilitating updates with minimal effort.

13 Operational and Environmental Requirements

13.1 Expected Physical Environment

Insert your content here.

13.2 Wider Environment Requirements

Insert your content here.

13.3 Requirements for Interfacing with Adjacent Systems

Insert your content here.

13.4 Productization Requirements

Insert your content here.

13.5 Release Requirements

Insert your content here.

14 Maintainability and Support Requirements

14.1 Maintenance Requirements

Insert your content here.

14.2 Supportability Requirements

Insert your content here.

14.3 Adaptability Requirements

Insert your content here.

15 Security Requirements

15.1 Access Requirements

Insert your content here.

15.2 Integrity Requirements

Insert your content here.

15.3 Privacy Requirements

Insert your content here.

15.4 Audit Requirements

Insert your content here.

15.5 Immunity Requirements

Insert your content here.

16 Cultural Requirements

16.1 Cultural Requirements

1. **Requirement:** The tool must avoid using colors 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 colors used in the tool are neutral and universally acceptable.
2. **Requirement:** The tool must support both metric and imperial measurement units.
    - **Rationale:** Users have different preferences and standards for measurement units. Supporting both metric and imperial units ensures accessibility and ease of use for a global audience.
    - **Fit Criteria:** Users can toggle between metric and imperial units for any measurements related to energy consumption.
3. **Requirement:** 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.

17 Compliance Requirements

17.1 Legal Requirements

Insert your content here.

17.2 Standards Compliance Requirements

Insert your content here.

18 Open Issues

- 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 increasing CPU and memory usage while running. The tool aims to optimize energy efficency 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 it's 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 working branch with format [main contributor name]/[descriptive topic]
- Commit changes with descriptive names
- Create unit tests for changes
- Create pull request to merge changes into epic branch
- Wait for all tests run with GitHub Actions to pass
- Wait for at least two approvals from teammates
- Merge changes into 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

Insert your content here.

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

Insert your content here.

23 Costs

Insert your content here.

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

Not Applicable.

25 Waiting Room

26 Ideas for Solution

Insert your content here.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

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

Mya Hussain Reflection

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