Hazard Analysis
Software Engineering

Team 4, EcoOptimizers

Nivetha Kuruparan
Sevhena Walker
Tanveer Brar
Mya Hussain
Ayushi Amin

Table 1: Revision History

Date	Developer(s)	Change
25 October 2024	All	Created initial revision of Hazard Analysis

Contents

1 Introduction

1.1 Problem Statement

The Information and Communications Technology (ICT) sector is currently responsible for approximately 2-4% of global CO_2 emissions, a figure projected to rise to 14% by 2040 without intervention (Belkhir and Elmeligi, 2018). To align with broader economic sustainability goals, the ICT industry must reduce its CO_2 emissions by 72% by 2040 (Freitag and Berners-Lee, 2021). Optimizing energy consumption in software systems is a complex task that cannot rely solely on software engineers, who often face strict deadlines and busy schedules. This creates a pressing need for supporting technologies that help automate this process. This project aims to develop a tool that applies automated refactoring techniques to optimize Python code for energy efficiency while preserving its original functionality.

1.2 Hazard Analysis Introduction

A hazard is defined as a property or condition in the system, combined with a condition in the environment, that has the potential to cause harm or damage—referred to as loss (Leveson, 2021). In software development, hazards can take various forms beyond just safety hazards, including security risks, usability challenges, incorrect inputs, or technical limitations like lack of internet connectivity.

This project focuses on developing an automated tool to refactor Python code for energy efficiency while preserving its original functionality. While this initiative holds significant potential for reducing CO_2 emissions in the Information and Communications Technology (ICT) sector, it also introduces various hazards. These hazards could arise from technical shortcomings, ethical challenges, or the inadvertent introduction of new problems during the refactoring process. This hazard analysis aims to identify and assess these risks to ensure the successful development and adoption of the tool.

2 Scope and Purpose of Hazard Analysis

The scope of this hazard analysis covers the potential risks and losses associated with the automated refactoring tool throughout its lifecycle. The primary hazards include:

- **Technical Failures**: Inaccurate refactorings, undetected code smells, or energy optimization that does not meet its intended goals could result in performance issues or loss of functionality.
- **Security Risks**: The automated nature of the tool may introduce security vulnerabilities, particularly if the refactorings unintentionally affect the security posture of the original code.
- **User Insensitivity**: If the tool is not designed with the users in mind, it could disrupt developer workflows or lead to the rejection of the tool. This can result in loss of productivity or missed opportunities for energy efficiency.
- **External Conditions**: The tool's dependency on environmental factors, such as the availability of internet connection or access to third-party libraries, could limit its usefulness in certain scenarios. This can lead to delays or failures in the refactoring process.

The purpose of this analysis is to identify these hazards, assess their potential impact, and outline strategies for mitigating them. By doing so, we aim to prevent losses related to time, resources, security, and the overall effectiveness of the tool, ensuring that it contributes positively to reducing the ICT sector's energy consumption and CO_2 emissions.

3 System Boundaries and Components

The system boundary refers to the library being developed and its core constituent modules, as well as peripheral tools that serve to expand on the system's utility and usability.

It is also important to make note of elements not controlled by the development team such as the database, the physical computer that will run the system, and any cloud hosting platforms used to utilize the system on a greater scale.

3.1 Core Modules

Energy Measurement Module

This module tracks and analyzes the energy consumption of the software being refactored, providing detailed metrics that help assess the efficiency of the code before and after refactoring. External tools will be used to implement this module with the primary one being pyJoules[1]. Other libraries may be added should the need arise.

Testing Module

The testing module runs automated tests on the refactored code to ensure that the functionality remains intact and that no errors are introduced during the refactoring process. Testing will be done through AST[2] parsing and any provided tests from the user.

Refactoring module

This core module identifies code smells and inefficiencies in the original code and suggests or applies appropriate refactorings to optimize the code for better performance and energy efficiency. Refactoring will be done through a mix of custom-made refactoring strategies and with the help of the python library, Rope.

Reinforcement Learning Model

The reinforcement learning model uses data from previous refactorings to improve its suggestions, helping the system learn and refine its refactoring strategies based on outcomes and energy consumption metrics. The model shall be built with the help of the machine learning library, PyTorch.

3.2 Peripherals

Visual Studio Code (VS Code) Extension

The VS Code extension provides a user-friendly interface within the IDE[3], allowing developers to interact with the refactoring tool, view energy consumption metrics, and apply refactorings suggestions directly from their development environment.

GitHub Action

The GitHub Action automates the refactoring process within CI/CD workflows, applying refactoring suggestions and running energy consumption analyzes during code integration, ensuring consistent energy-efficient practices.

Web Client

The web client offers a user interface that allows users to interact with the refactoring system remotely, enabling them to view energy consumption reports, and track performance metrics from a browser.

[1] A python library that energies the energy footprint of a host machine.
[2] Abstract Syntax Tree: a data structure used to represent the structure of a program.
[3] Integrated Development Environment

4 Critical Assumptions

- The Energy Measurement Model will provide accurate and consistent energy consumption metrics across different platforms (Windows, macOS, Linux). There are no discrepancies in measurements due to platform differences that could result in ineffective refactoring.
- The Testing Module is provided with automated tests that have enough coverage to detect post refactoring bugs, functionality regressions, etc.
- Code smells identified by the Refactoring Module always involve code that could be more energy efficient.
- Custom-made refactoring strategies and Rope are capable of generating effective and correct refactoring.
- Sufficient data sets are available for the reinforcement learning model to provide increasingly accurate and efficient refactoring suggestions over time.
- GitHub Actions, which is a third-party dependency for the DevOps integration, is not suspended for a prolonged period of time.

5 Failure Mode and Effect Analysis

6 Failure Mode and Effect Analysis

Table 2: FMEA Table

Component	Failure Modes	Effects of Failure	Causes of Failure	Recommended Action	SR	Ref
Energy Measurement	Background tasks could be incorrectly included in energy measurement.	Background tasks that are not to the Python code under refactoring could skew the overall result for consumed energy. This could: • skew the energy consumption metrics and mislead users. • produces refactorings that do not save energy due to faulty measurement.	The Energy Measurement Module lacks a filtering mechanism to isolate the specific Python code snippet being refactored. This allows unrelated background tasks or idle processes to be included in the overall energy measurement.	Use process-level tracking to distinguish between the Python code under refactoring and unrelated background tasks.	SCR-1	HZ 1

Table continues on next page

Table 2: FMEA Table

Component	Failure Modes	Effects of Failure	Causes of Failure	Recommended Action	SR	Ref
Energy Measurement	The Energy Measurement Module does not provide energy consumption data in a timely manner	• User experiences delays in receiving energy consumption feedback, which can slow down their refactoring process. • The tool may be considered inefficient by users, potentially causing them to not adopt it.	• Computational overhead in the Energy Measurement Module • Delays in accessing low level hardware components that are needed for energy measurement	• Investigate PyJoules' configuration options to find a balance between accuracy and performance based on the size and complexity of the code being refactored. • Implement parallel processing to measure energy consumption and run code smell detection simultaneously. This can reduce the overall time by allowing energy measurements to be done without holding up other tasks. • Implement a graceful timeout mechanism if PyJoules takes too long to respond. • Provide users with an estimated time for completion so they are aware of ongoing measurements if energy measurement exceeds a set time.	SCR-1, SCR-10	HZ 2

Table continues on next page

Table 2: FMEA Table

Component	Failure Modes	Effects of Failure	Causes of Failure	Recommended Action	SR	Ref
Energy Measurement	The energy measure module does not provide any data at all	Refactoring fails due to no energy metrics available for validation of changes	• The system does not have the necessary administrative or system-level permissions to access energy-related data, especially in cloud environments • The energy measurement process might be too slow, resulting in timeouts or delays that cause no metrics to be reported within the expected time frame.	• Ensure the software has sufficient permissions to access low-level system metrics, such as power usage, and grant administrative privileges if needed. • Increase the allowed time frame for measurements to complete • Implement a functionality in the system that allows that prompts the user with a request to pause the refactoring process and restart at the same point when the system is less busy	SCR-1, SCR-3, SCR-10	HZ 3
	text					HZ 4
Testing	Test not able to run due to refactoring	• Testing coverage not met • Unable to test business logic of user code • Unable to complete refactorings	Test cases dependent on some modules that have been refactored	• Use AST as a base for testing • Ensure that any refactorings that involve variable, class or function name changes are disabled on default and require explicit enabling from the user	SCR-2	HZ 5

Table continues on next page

Table 2: FMEA Table

Component	Failure Modes	Effects of Failure	Causes of Failure	Recommended Action	SR	Ref
Testing	Provided test suite misses critical scenarios	• The refactored code could fail in production under specific conditions, leading to potential downtime or incorrect behaviour.	• Limited test suite or lack of coverage for particular scenarios.	Implement syntactical analysis in refactoring to mitigate code functionality changes	SCR-2	HZ 6
Refactoring	Incorrect refactorings suggestions were given	• Refactored code increases the energy consumption instead of reducing it. • Functionality of refactored code is not consistent from that of the original code.	• Inadequate training of the reinforcement learning model. • Refactoring logic misses some edge cases. • Reinforcement learning model creates syntactically incorrect code.	Validate the changes by verifying energy consumption statistics before applying changes to the code by adding validation rules	SCR-2	HZ 7
	A memory leak occurs during the refactoring process	• Gradual increase in memory usage leading to application lagging, crashing or freezing	• Poor memory management during the refactoring process	Implement automatic garbage collection or memory de-allocation after each refactoring step	SCR-8	HZ 8
	Unable to revert refactorings	• User loses confidence in integrity of system • Unable to pick which refactorings to keep and which to discard based on user input	• Faulty version control strategy	Implement a robust version control system that follows a granular commit system tracking each change with precision	SCR-4	HZ 9

Table continues on next page

Table 2: FMEA Table

Component	Failure Modes	Effects of Failure	Causes of Failure	Recommended Action	SR	Ref
Refactoring	The refactoring improves energy efficiency but degrades other performance metrics like speed or memory usage	• The software becomes slower or uses more memory, which could counteract the benefits of energy optimization.	• Poor trade-offs made by the refactoring algorithm between energy efficiency and other performance factors.	Implement multifactor optimization, balancing energy efficiency with other performance metrics. If this is not possible inform the user of potential degradation when suggesting at-risk refactorings.	SCR-2	HZ 10
	The refactoring tool modifies code that relies on external libraries, causing incompatibility with these libraries.	• Code fails to execute or produces unexpected behaviour due to altered interactions with third-party libraries.	• Lack of awareness of how certain refactorings impact external dependencies, especially with complex or dynamically loaded libraries.	Implement a detection mechanism that identifies external library dependencies and exempts them from refactorings unless explicitly requested by the user.	SCR-5	HZ 11
	The tool accesses or refactors code that contains sensitive information (e.g., API keys, credentials), which could lead to unintentional exposure or mismanagement of this data.	• Sensitive information could be mishandled, leading to potential security breaches, privacy violations, or unauthorized access.	• Refactorings alter or expose parts of the code that store or transmit sensitive data, without proper checks.	Implement security-focused static analysis tools that identify sensitive code sections and prevent them from being refactored. Warn users when refactoring such areas.	SCR-6	HZ 12
Reinforcement Learning	Model overfitting	Less effective when applied to unseen or more diverse codebases resulting in suboptimal and/or incorrect refactorings for new projects.	• Over-training model on similar datasets	Use a diverse and representative dataset for training the model	SCR-7	HZ 13

Table continues on next page

Table 2: FMEA Table

Component	Failure Modes	Effects of Failure	Causes of Failure	Recommended Action	SR	Ref
Reinforcement Learning	Bias in recommendations	Model starts favouring certain types of refactorings or ignoring others that could be more efficient for different scenarios.	• Imbalanced reward function • Not enough exploration actions done by the model • Unrealistic straining data simulations used	• Regularly audit the model for bias • Ensure the training data is balanced across different types of refactorings and code patterns. • Regularly retrain the model	SCR-7	HZ 14
	Model drift and degradation	The RL model becomes less effective over time due to changes in code styles, and best practices, or the introduction of new refactoring strategies leading to a degradation in performance and accuracy of refactoring suggestions.	Passing of time and evolution of software practices	• Regularly retrain the model using up-to-date data and monitor its performance to detect signs of drift. • Implement a feedback loop to incorporate user corrections into the training data.	SCR-7	HZ 15
	Over-reliance on pre-trained models. The reinforcement learning model is overly trusted, even when it generates suboptimal or erroneous refactorings.	• Incorrect or harmful refactorings are applied without proper oversight, leading to system instability.	• Over-reliance on automated suggestions without sufficient human review or fail-safes.	Require human approval for significant refactorings or apply thresholds to reject low-confidence suggestions from the model.	SCR-9	HZ 16

7 Safety and Security Requirements

SCR 1. *The system shall log all energy consumption measurements with timestamps and indicate which processes were measured to aid in future analysis and troubleshooting.*
Rationale: Detailed logging with timestamps and process attribution ensures accurate energy data and helps identify delays or misattributions.
Fit Criterion: 100% of energy analysis logs must include timestamps and process-level breakdowns of all measured processes.
Associated Hazards: HZ-1, HZ-2, HZ-3
Priority: High

SCR 2. *The system shall ensure that all refactored code has comprehensive test coverage and passes performance metrics such as energy efficiency, speed, and memory usage.*
Rationale: Proper test coverage and performance checks prevent faulty code from being introduced and ensure refactorings improve or maintain performance.
Fit Criterion: 100% of refactorings must pass tests covering all code paths, and performance must remain within a 5% tolerance across energy, speed, and memory metrics.
Associated Hazards: HZ-4, HZ-5, HZ-6, HZ-9
Priority: High

SCR 3. *The system shall check for necessary system-level permissions to access energy consumption data and alert users if permissions are missing.*
Rationale: Lack of access may lead to failure in energy data retrieval, which can hinder the accuracy of analysis.
Fit Criterion: 100% of runs shall check for and request permissions if required, and alert the user in case of failures.
Associated Hazards: HZ-3
Priority: High

SCR 4. *The system shall ensure version control for each refactoring, allowing changes to be reverted in case of errors.*
Rationale: Version control helps prevent loss of code or data and allows developers to revert refactorings if necessary.
Fit Criterion: 100% of changes shall be recorded, allowing full reversion with no data loss.
Associated Hazards: HZ-8
Priority: High

SCR 5. *The system shall detect and exempt external library dependencies from refactorings to avoid compatibility issues.*
Rationale: Modifying external dependencies could lead to system instability or incompatibility with other tools or frameworks.
Fit Criterion: 100% detection accuracy for external library code during refactoring.
Associated Hazards: HZ-10
Priority: Medium

SCR 6. *The system shall not refactor or alter code containing sensitive information (noted by user), ensuring security is maintained.*
Associated Rationale: Refactoring sensitive code may introduce vulnerabilities and compromise security.
Fit Criterion: 100% of refactorings must pass a security check to avoid tampering with sensitive information.
Associated Hazards: HZ-11
Priority: High

SCR 7. *The reinforcement learning model shall be trained on diverse datasets and periodically audited to avoid bias and prevent degradation.*

Rationale: Overfitting or model degradation can lead to suboptimal or biased refactorings, impacting the system's effectiveness.
Fit Criterion: 95% of refactorings should be equally effective across different types of projects, and model audits should occur at least quarterly.
Associated Hazards: HZ-12, HZ-13, HZ-14
Priority: Medium

SCR 8. *The system shall implement memory leak detection during refactoring and alert users if any issues are detected.*
Rationale: Memory leaks may cause system crashes and reduce performance.
Fit Criterion: 100% of memory leak incidents should trigger an error alert and resolution process.
Associated Hazards: HZ-7
Priority: Medium

SCR 9. *The system shall require user approval for high-impact refactorings or those with low confidence, providing visibility and oversight for critical changes.*
Rationale: Automated decisions could introduce errors without human oversight, and users should be aware of significant changes.
Fit Criterion: 100% of high-risk or low-confidence refactorings must require user approval before proceeding.
Associated Hazards: HZ-15
Priority: High

SCR 10. *The system shall alert users to any delays or failures in reporting energy consumption, ensuring transparency in reporting.*
Rationale: Users need to be aware of any issues in energy reporting to troubleshoot and resolve potential problems.
Fit Criterion: 100% of energy measurement delays or failures must trigger a user alert.
Associated Hazards: HZ-2, HZ-3
Priority: High

8 Roadmap

Requirements that will be implemented during the capstone timeline:

- SCR 1
- SCR 2
- SCR 3
- SCR 4
- SCR 5
- SCR 6
- SCR 9
- SCR 10

Requirements implemented in the future:

- SCR 7: This will be audited on a regular basis which will be a future implementation.
- SCR 8: This can be implemented in the future as it is not a high priority and not the biggest concern to this project.

Appendix — Reflection

Nivetha Kuruparan

1. *What went well while writing this deliverable?*

   While writing this hazard analysis, one thing that went well was identifying missing requirements that had not been captured in the original SRS document. As we analyzed potential hazards, especially related to security and communication, it became clear that certain protections—like secure authentication and making sure we are tracking the correct energy needed more attention. Catching these gaps allowed us to enhance the system's robustness and ensure that our requirements addressed both safety and security concerns comprehensively. This process also helped align our priorities more effectively, as we were able to associate risks with specific requirements and refine the overall design.

2. *What pain points did you experience during this deliverable, and how did you resolve them?*

   A pain point during this deliverable was mapping out the safety requirements to the identified hazards. It was challenging to ensure that each requirement accurately addressed specific risks, especially when certain hazards overlapped or required more nuanced handling. Determining the exact scope of each safety requirement, while avoiding redundancy, took considerable time and effort. To resolve this, we revisited the hazard analysis step-by-step, carefully analyzing each potential failure and its impact on the system, which helped clarify how the requirements should be structured. Collaborating with the team to cross-check each hazard also helped ensure that we didn't overlook critical risks or assign incorrect priorities.

Sevhena Walker

1. *What went well while writing this deliverable?*

   One thing that went really well during the hazard analysis was how it helped me catch issues I'd originally missed. The structured process made it easier to step back and look at our project from a different perspective, which helped highlight potential risks I hadn't thought of before.

2. *What pain points did you experience during this deliverable, and how did you resolve them?*

   I'll be honest the worst part of this deliverable was formatting the FMEA table in latex. It doesn't seem right to talk about pain points without mentioning the one thing that truly had me pulling my hair out. In terms of the actual content of the deliverable, brainstorming hazards was challenging, but not exactly a pain. The challenging part was coming up with solution or mitigating actions to counter those hazards. Some components, like the reinforcement model, I have truly no experience with and its pretty hard to come up with solutions to risks you have never even experienced, let alone thought of.

Tanveer Brar

1. *What went well while writing this deliverable?*

   This deliverable was pretty short compared to previous ones but we were still on top of our toes when it came to planning it within the team. I like that we allowed everyone to pick up topics that interested them the most and left the key piece of work(FMEA table) to be worked on collaboratively in Overleaf by everyone. Timely spiliting of the work gave us ample time to finish individual assignments as well as review other people's contributions.

2. *What pain points did you experience during this deliverable, and how did you resolve them?*

   The main challenge that I faced was mapping the Failure Modes to appropriate security requirements. Some of the failure modes that I came up with aligned with the security requirements written previously, but more content needed to be added to those. To resolve this, I added the additional description needed for these security requirements for some hazards and created new requirements for others.

Mya Hussain

1. *What went well while writing this deliverable?*

   We divided up the work early and were all able to complete sections at our own pace or ahead of time depending on our midterm schedules. This week was particularly busy because all of us had midterms so I was able to complete my section during reading week to reduce the capstone workload during the week. Although I will say it's a little disappointing that every time we are done on time (which has been every time so far) the deliverable is extended last minute. I don't want to complain too much though because I have a feeling that if I do complain it won't be extended next time I'd actually like it to be. So far team dynamics and morale have been good. I appreciate the level of organization we've been able to have so far as it made collaborating so much smoother and has helped everyone stay on track with our tasks.

2. *What pain points did you experience during this deliverable, and how did you resolve them?*

   Determining which factors qualify as hazards for our analysis was somewhat unclear. A hazard is defined as anything with the potential to cause harm or loss, yet certain risks may emerge from poor design, complicating our decision on whether to include them. For example, user interface hazards like "the tool does not provide clear feedback to the user after refactoring" can technically be classified as a hazard. While we aim to mitigate team-imposed hazards, it raises the question of whether we should simply avoid designing a flawed product in the first place, and not include these hazards in the analysis or if we should do a worst-case analysis and include every possible pitfall. The same argument could be made for some security hazards for example "while parsing user input code, the software encounters malware and executes it," avoiding this is something a good tool should already have built in, so it begs the question of "how bad do we envision our final product when analyzing hazards?" We were able to get some clarification on this in our TA 1-1 meeting but ultimately tried to keep it high level so our report didn't end up being too long.

Ayushi Amin

1. *What went well while writing this deliverable?*

   I think one of the best things about writing this deliverable was how well we collaborated using Overleaf. It made it super easy to work together on the FMEA table. We divided up the work, so everyone had their own sections to focus on, but we also helped each other out when needed. This teamwork really made a difference because we could share ideas and give feedback in real time. Even though we had midterms this week, which delayed our progress a bit, everything ended up working out. We managed our time well, and I was impressed with how we all stayed on track despite the busy schedule. It felt good to see how our combined efforts came together in the final product. Overall, I think our collaboration really strengthened the quality of our work.

2. *What pain points did you experience during this deliverable, and how did you resolve them?*

   One big challenge I faced was figuring out the difference between general risks and specific hazards for our project. At first, it was a bit confusing, and we spent some time debating whether certain issues were specific enough. To resolve this, I looked up examples from other projects, which helped clarify things for everyone. Overall, even though there were some bumps along the way, working through these challenges taught me a lot about hazard analysis and teamwork in software development.

Group Answer

3. *Which of your listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? For the latter ones (ones you thought of while doing the Hazard Analysis), how did they come about?*

   The risks that we had thought of before this deliverable include HZ6, HZ7, HZ9 and HZ10. All remaining risks(HZ1, HZ2, HZ3, HZ4, HZ5, HZ8, HZ11, HZ12, HZ13, HZ14, HZ15 and HZ16) were thought of during the deliverable. To come up with ideas, we analyzed the system on a component by component basis in order to identify risks on a granular level(components defined earlier in Section 3 of this document). Defining critical assumptions before brainstorming the risks helped create a boundary for lookout for possible things that could go wrong with each component. It is important to note that a deeper understanding of our dependencies, such as PyJoules for energy measurements, helped identify possible things that could go wrong when implementing those in their respective modules. We adopted an iterative approach to the brainstorming, as identifying ground level risks helped to identify other risks over an entire week of deliberation.

4. *Other than the risk of physical harm (some projects may not have any appreciable risks of this form), list at least 2 other types of risk in software products. Why are they important to consider?*

   (a) Data Security Risk: Software products are a storehouse of data related to its users. If sensitive data is exposed to vulnerabilities, it can leads to breaches. This risk is important to consider as a a breach can harm users as well as the organization's reputation. Addressing this risk is critical for maintaining trust and preventing legal battles.

   (b) Operational Risk: Live hosted software products are bound to face risks related to live performance, such as slow performance and/or system downtime. These are important to consider as they are post-implementation risks that directly impact system availability to users. This can impact user productivity and cause financial loss to the organization, which is why they should be considered.

References

Lotfi Belkhir and Ahmed Elmeligi. Assessing ict global emissions footprint: Trends to 2040 and recommendations. *Journal of Cleaner Production*, 2018. URL https://www.researchgate.net/publication/322205565_Assessing_ICT_global_emissions_footprint_Trends_to_2040_recommendations#:~:text=in%20industrial%20development.-,...,%25%20by%202040%20%5B11%5D%20.

Charlotte Freitag and Mike Berners-Lee. The real climate and transformative impact of ict: A critique of estimates, trends, and regulations. *Patterns Volume 2, Issue 9*, 2, 2021. doi: 10.1016/j.patter.2021.100340. URL https://www.sciencedirect.com/science/article/pii/S2666389921001884#:~:text=If%20the%20ICT%20sector%20should,these%20sectors%20will%20have%20to.

Nancy Leveson. How to perform hazard analysis on a 'system-of-systems'. *Massachusetts Institute of Technology*, 2021. URL http://sunnyday.mit.edu/SOS-hazard-analysis.pdf. Accessed: 2024-10-16.