Microwave Defrosting Program

Final Review for Software Engineering Project

1. Introduction

The goal of this project was the agile, incremental development of a microwave defrosting application simulating an industrial real-world scenario. The project demonstrates iterative software engineering phases from requirement analysis to testing, with a focus on modularity and reusability.
The complete class diagram can be viewed here.

2. Assignment Mapping to Developed Product

The formal declaration of independent work, including the development environment details, has been submitted as a separate document.

The project focuses on five key part-functionalities:

- Door-open interruption
- Overheating prevention
- Power control (expanded to power and cooling control)
- Control of radiation distribution
- Interior lighting, replaced by an error alarming feature to notify users of faults or unsafe states

The reviewer has access to all relevant project resources:

- Codebase: GitHub Repository
- Wiki (requirements, architecture, design, iterations, testing): GitHub Wiki
- Project management board: GitHub Projects
- Test run documentation: GitHub Actions

3. Assignment–Implementation Mapping Questions

Formal Criteria & Documentation

- **Declaration of Original Work:**
  A signed declaration of independent work was submitted as a separate document, including the development environment.

- **Part-functionalities Named in the Declaration:**
  The five main part-functionalities were clearly defined and listed as required.

Development Requirements

- **Development Environment Access:**
  Access to the complete development environment is provided via the GitHub links listed above.

- **Presentation:**
  Preparation for the post-submission presentation is planned according to the examiner's schedule.

Software Engineering Phases

- **Requirement Engineering:**
  Requirements were documented and traced in detail in the wiki: Requirements Engineering
  A traceability matrix links requirements to implemented features and tests: Traceability Matrix.

- **Software Architecture:**
  The system structure and module responsibilities are documented using markdown and UML diagrams in the wiki: Software Architecture.

- **Software Design:**
  Design artifacts such as sequence and component diagrams were created to model behavior and interactions. These are documented in: Software Design.

- **Implementation:**
  The software was developed iteratively with at least three working software versions demonstrated. Detailed iteration protocols are available at: Implementation & Iterations.

- **Software Testing:**
  Testing covers unit and integration levels, documented with traceability to requirements here: Testing & Quality Assurance.

Project Constraints

- **Simulated Interfaces:**
  Hardware interactions such as sensors were mathematically simulated. User interaction was modeled by a predefined start sequence due to lack of physical hardware.

- **Reuse of Existing Solutions:**
  Frequent libraries and modular design were reused following real-world manufacturer recommendations. External components are marked and explained in the documentation.

4. Implementation Steps

The development proceeded in five main steps across three iterations:

1. Initial setup, planning, and documentation (no iteration) to establish project scope and environment.
2. First iteration focused on implementing the system control loop and the core backbone of the application.
3. Second iteration addressed all safety-critical components, ensuring security measures and system axioms that provide a stable foundation for subsequent development.
4. Third iteration completed the remaining essential components, overall program control, and the defrosting functionality.
5. Additional development beyond the core requirements was pursued due to interest and a comfortable development workflow, including modularization improvements, enhanced simulations, and better documentation to facilitate future reuse.

5. Challenges

The project began with a loosely defined task, making it difficult to understand exact requirements and objectives initially. This lack of clarity complicated early planning and slowed development. As my first solo software engineering project, I struggled to accurately estimate workload, leading to both over- and under-allocation of effort.

The documentation process was especially demanding, requiring detailed, rigorous work that was challenging to maintain alone. Balancing thorough documentation with ongoing development was a continuous challenge.

Iterative development cycles were unevenly paced, resulting in fluctuating workloads—some phases were manageable, while others demanded extensive effort.

Finally, testing all components thoroughly was complex and time-consuming, as ensuring full traceability and compliance with requirements added difficulty.

6. Successes

A fully functional product was delivered successfully. Extensive upfront planning and documentation eased implementation, enabling smooth progress after initial groundwork. Although the planning phase felt like overhead, establishing a steady workflow made development rewarding.

Seeing the complete system operate and simulate its behavior as intended brought pride and motivation. The project highlighted the importance of following the full software engineering process—from requirements to testing—and reinforced the value of preparation and iterative development in delivering reliable software.

7. Left Out Parts

Not all planned features are fully operational; only essential components are complete to ensure a working product. Some parts may be implemented before the final submission.

Currently, during idle phases, the turntable continues spinning as a basic simulation. Interface comments and documentation have begun but remain incomplete, as I prioritized finishing core features first to avoid burnout. Given expected changes, detailed commenting was postponed to maintain momentum.

Overall, while foundational work is solid, further improvements are planned.

8. Lessons Learned / Improvements

Interface planning was a helpful alignment tool to clarify expectations and coordinate development but did not fully represent the final product's complexity and evolving requirements. The initially defined part-functionalities provided a good starting point, but real-world projects required flexibility as demands changed.

I faced several challenges balancing thorough documentation with practical progress. Designing a clear documentation structure was difficult, and I sometimes followed inefficient paths that slowed progress. These experiences emphasized the need for iterative refinement in code, project management, and documentation. These lessons will guide future projects to better manage scope, documentation, and evolving requirements.