Milestone Two Narrative

Category: Software Design and Engineering  
Artifact: Raspberry Pi Smart Thermostat  
Student: Joshua Emmons

🔧 Artifact Description

The artifact I have chosen to enhance is a Raspberry Pi–based smart thermostat system originally developed in CS 350. This system effectively regulates heating and cooling operations through an intuitive interface comprising GPIO buttons, a character LCD screen, and temperature/humidity data sourced from an Adafruit AHT20 sensor.

However, the true sophistication lies within the logic. This system operates using a finite state machine (FSM) that governs state transitions among Off, Heating, and Cooling based on real-time data. With visual LED feedback, UART-based serial communication, and a tactile physical interface, the thermostat approximates an element from a smart home laboratory.

Since its inception, I have transformed this project from a mere functional prototype into a modular, scalable, and robust system. Enhancements implemented during CS 499 have advanced the project closer to a production-grade embedded design, while simultaneously establishing a foundation for future Internet of Things (IoT) integration.

✅ Justification for Inclusion in ePortfolio

I selected this artifact because it represents the perfect intersection of my skills in embedded hardware, Python development, and software design principles. It’s an actual hybrid project—part engineering, part software architecture—and it allowed me to demonstrate real-world capabilities beyond academic assignments.

Enhancements that showcase this include:

🧱 Modular Architecture: Refactored code across distinct files for clarity and scalability: utils.py, constants.py, data\_logger.py, and setpoint\_storage.py.

🔁 Robust FSM Design: A structured FSM using the python-statemachine library ensures predictable and maintainable control flow.

📜 Config-Driven Behavior: System settings (poll intervals, LED modes, default set point) are now pulled from a config.json file, allowing easy adaptation without code changes.

📀 Persistent State Management: The current set point is saved and reloaded between restarts, mimicking the behavior of commercial thermostats.

📈 CSV Logging: The system logs every temperature and state change to a readable .csv file for later analysis.

🚨 Fail-Safe Feedback: Graceful error handling displays messages like "Sensor Error" when faults are detected.

💡 Rich Documentation: Every module includes meaningful comments and docstrings, ensuring maintainability for future developers (or future me).

This artifact shows that I’m not just capable of writing scripts—I can design, document, and enhance systems that interact with real-world hardware under constraints.

📊 Outcome Alignment

This project aligns strongly with the following CS program outcomes:

|  |  |
| --- | --- |
| Program Outcome | Demonstrated By |
| Software Engineering & Design | Modularization, config-driven behavior, LED abstraction, and FSM design. |
| Data Structures & Algorithms | State machine logic and real-time condition-based transitions |
| Security Mindset | Graceful sensor failure handling, input validation, persistent state safety |
| Technical Communication | Inline documentation, meaningful LCD messaging, updated README, and flowcharts. |

I’ve met my original coverage plan, and with future enhancements, I’ll continue deepening alignment with these outcomes.

🌟 Future Enhancements and Expansion Opportunities

While the current version is feature-rich and stable, I’m just getting started. Here’s what’s on the roadmap:

🌫️ Humidity Monitoring: Since the AHT20 already supports relative humidity, I plan to add humidity data to the LCD and CSV logs. This unlocks future actions, such as automated dehumidifier control or a comfort index display.

🌐 Web Dashboard (Flask or FastAPI): A lightweight browser-based interface where users can view current readings, adjust the set point, and view live logs from their phone or laptop.

🗓 Time-Based Set Point Scheduling: Integrating a JSON-based schedule will allow users to define dynamic temperature targets throughout the day, just like a Nest.

🛡️ Remote Telemetry via MQTT or HTTP: Sending sensor data to a cloud broker or dashboard would allow remote monitoring, intelligent alerts, and cross-platform integration (e.g., Home Assistant).

📉 Graphical Log Visualization: By charting the CSV log data with Plotly or Matplotlib, I can create visual insights into temperature stability and efficiency.

↺ Watchdog System: A built-in watchdog thread can detect when the sensor fails or the display thread hangs, triggering fallback behaviors such as blinking LEDs or rebooting the service.

🤻 Dockerization & CI: Packaging the entire system in a Docker container with mocked GPIO interfaces and unit tests will enable CI/CD pipelines and development without hardware.

Each of these would further transform the thermostat from a cool class project into a professional-grade embedded system.

🪞 Reflection on the Enhancement Journey

When I began enhancing this artifact, my primary challenge was untangling the original monolithic script and introducing clear boundaries between its components. It forced me to ask hard questions: How should sensor logic be abstracted? What if the sensor fails? What if I want to test this on a laptop with no GPIO?

Breaking things apart and putting them back together again taught me:

🧠 How to write modular Python code that scales across hardware platforms

📆 How to use config files and persistence to emulate real-world embedded system behavior

🧪 How to build testable, fault-tolerant embedded logic

📡 How logging and serial communication empower diagnostics and future connectivity

Most importantly, this enhancement process deepened my confidence in designing embedded systems that not only work but are also maintainable, secure, and expandable.

I’m proud to include this project in my professional portfolio because it reflects more than just code—it reflects growth, vision, and the readiness to tackle real-world engineering problems.