Final Report: Development Process, Git Usage, Testing Practices, and Challenges

Development Process

The development of the temperature sensor simulation application was a multi-phase process that involved both technical and strategic decisions. The primary objective was to create a console application that could simulate temperature readings from a virtual data centre sensor, validate these readings, detect anomalies, and store historical data.

The first step in the development process was planning and designing the core features. I began by breaking the application down into smaller, manageable components. These included initializing the sensor with configuration data, simulating temperature readings with noise and variability, validating data against acceptable ranges, detecting anomalies, and storing the data in a database. Additionally, logging and fault injection were incorporated to simulate real-world environments where sensor failures could occur, and to help with debugging and analysis.

Once the high-level design was established, I proceeded to implement each feature in isolation, ensuring that the application was modular and extensible. This approach allowed me to focus on the logic of each individual feature before integrating them into a cohesive whole.

Git Usage

Git was an essential tool throughout the development process. It allowed me to manage versions of the code, track changes, and collaborate effectively. From the start, I created a repository and established a basic branching strategy where I worked on individual features in separate branches, merging them into the main branch after completing each feature.

Commit Frequency and Messages: I committed code frequently, typically after completing a feature or fixing a bug. This allowed me to maintain a clean version history, making it easier to trace changes. For example, after adding sensor initialization, I committed with a message like: “Implemented sensor initialization from config file.” This practice helped ensure that the development process was transparent and allowed for easy rollback if needed.

Handling Merge Conflicts: While Git significantly improved my workflow, there were instances of merge conflicts when two branches modified the same files. I resolved these conflicts by reviewing the changes carefully and selecting the correct approach. This forced me to carefully think through the integration of different features and ensured I didn’t miss any logic changes.

Git Ignore: I configured a `.gitignore` file early in the process, ensuring that temporary files (such as the `.vs` folder created by Visual Studio) were excluded from the repository. This step helped prevent unnecessary files from being tracked and cluttering the repository.

Testing Practices

Testing was an integral part of the development process, as it ensured that the application met its requirements and functioned as expected. I approached testing in two phases: unit testing and integration testing.

Unit Testing: Each core feature of the application was unit tested to verify its functionality in isolation. For example, the `SimulateData` method was tested by generating random readings and ensuring they fell within the specified range, with added noise. I also created unit tests for the `ValidateData` method to ensure that it correctly validated readings against the acceptable temperature range.

Integration Testing: Once individual components were tested, I began integrating them. I performed integration testing to ensure that the entire system worked together as expected. For instance, I tested the flow of data from sensor initialization to data simulation, validation, logging, and anomaly detection. I used test data to simulate different scenarios, such as valid temperature readings and sensor failures, to ensure the system could handle various conditions without errors.

Logging and Anomaly Detection Testing: Testing the logging system was crucial to ensure that data was properly recorded and anomalies were detected. I simulated both valid and invalid temperature readings and checked that they were logged correctly, while also verifying that anomalies were detected when readings deviated significantly from the expected range.

Challenges Faced

During development, I encountered several challenges:

1. **Data Validation Logic:** The most significant challenge was ensuring that the data validation worked correctly in all cases. I had to account for edge cases where temperature readings might fall just outside the acceptable range, and I needed to handle these situations gracefully.
2. **Simulating Faults and Anomalies:** Simulating faults and anomalies, such as sensor failures and sudden temperature spikes, required creating random fault conditions in the sensor data. This was challenging as I had to ensure that these faults were injected at appropriate intervals and that they didn't overwhelm the application's normal behavior.
3. **Git Merge Conflicts:** As mentioned, managing merge conflicts when integrating features was a challenge. At times, the conflicts arose from parallel changes made

to shared files, especially when modifying configuration-related code. This required careful analysis to ensure the proper resolution of conflicts.

4. **Permission Issues with Git:** I faced issues with file permissions when Git attempted to add files in the `.vs` folder. This was resolved by adding the `.vs` folder to the `.gitignore` file and using `git rm` to untrack any previously committed files.

Conclusion

Overall, the development of the temperature sensor simulation application was a rewarding learning experience. By following an organized approach to feature development, using Git for version control, and rigorously testing the application, I was able to create a reliable and functional system. The challenges faced, particularly around data validation and fault injection, provided valuable insight into the complexities of real-world systems and the importance of robust error handling and testing practices.