Analysis/Design Phase

SWEN326 Design Document

Method:

- Power of ten rules
- Good code commenting
- Unit tests etc…

Will it be responsive (resize components):

Window should either be responsive, or we should agree on a size.

Get a window working and then choose as a group the window size. The window size should be small enough to fit on a 13-inch laptop.

Alternatively, we can make the window responsive (be able to resize to any size), but this will add additional work and complexity to the code.

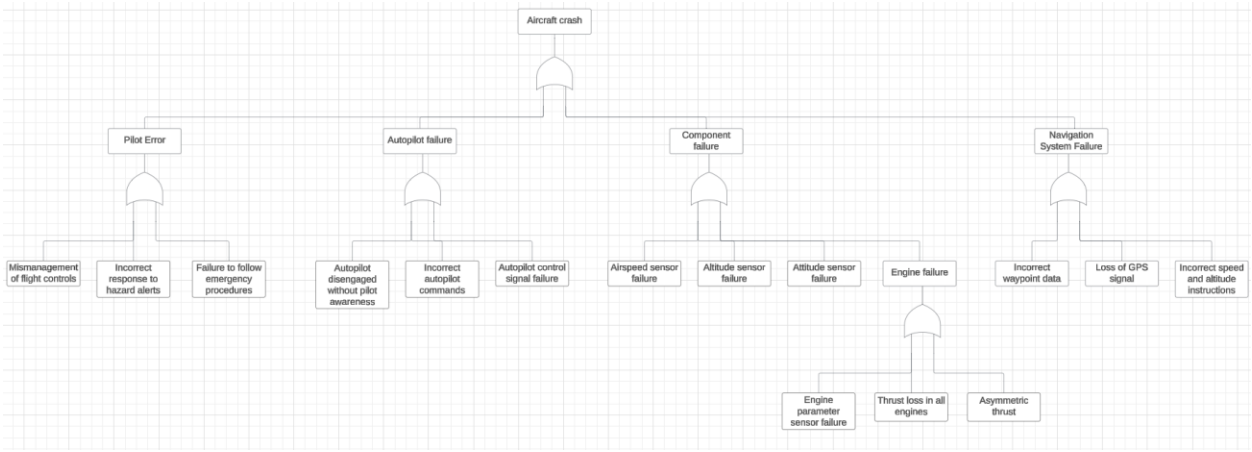
Design:

- Start by writing the main method and the code setting up the JavaFX application.
- Modules list:

- Control panel
- Simulator
- Main Application
- Testing

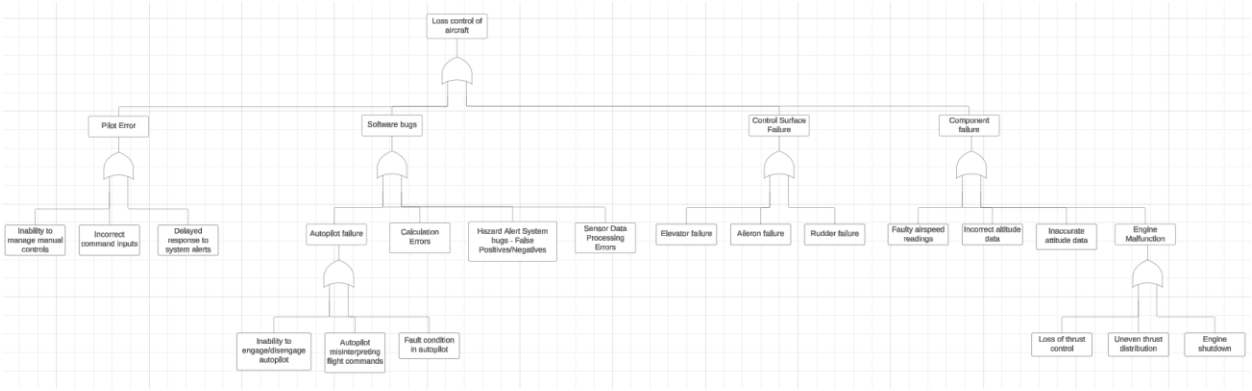
Fault tree analysis:

Aircraft crash:



https://lucid.app/lucidchart/6360d839-2cdb-40b6-9f90-62cc67fbbf71/edit?viewport_loc=-1119%2C-168%2C4616%2C1938%2C0_0&invitationId=inv_b8b8281c-505d-4933-b980-0b3563c9974e

Loss control of aircraft:



https://lucid.app/lucidchart/e8ef2a35-0dc9-46f9-adc4-8e1f87d9101c/edit?viewport_loc=-1125%2C-233%2C5637%2C2367%2C0_0&invitationId=inv_34594e65-f31f-4352-a126-9634afc57b52

Explanation and Mitigation:

Pilot Error:

Ensure thorough pilot training and regular simulations.

Implement strict adherence to checklists and emergency protocols.

Implement bright flashing warning lights in the console.

Autopilot System Failure:

Regular maintenance, system checks and simulations.

Implement redundancy in autopilot control systems.

Comprehensive testing of autopilot software under various scenarios.

Sensor Failure:

Use 2oo3 architecture for critical sensors to ensure reliability.

Implement fault detection and tolerance mechanisms.

Engine Failure:

Regular engine maintenance and inspection.

Implement engine redundancy where possible.

Navigation System Failure:

Ensure robust and redundant navigation systems.

Regularly update and validate waypoint data.

Software bugs:

Rigorous testing and validation

Implementation of error-checking mechanisms for data integrity

JavaFX Strategy:

Directly in Java and not FXML.

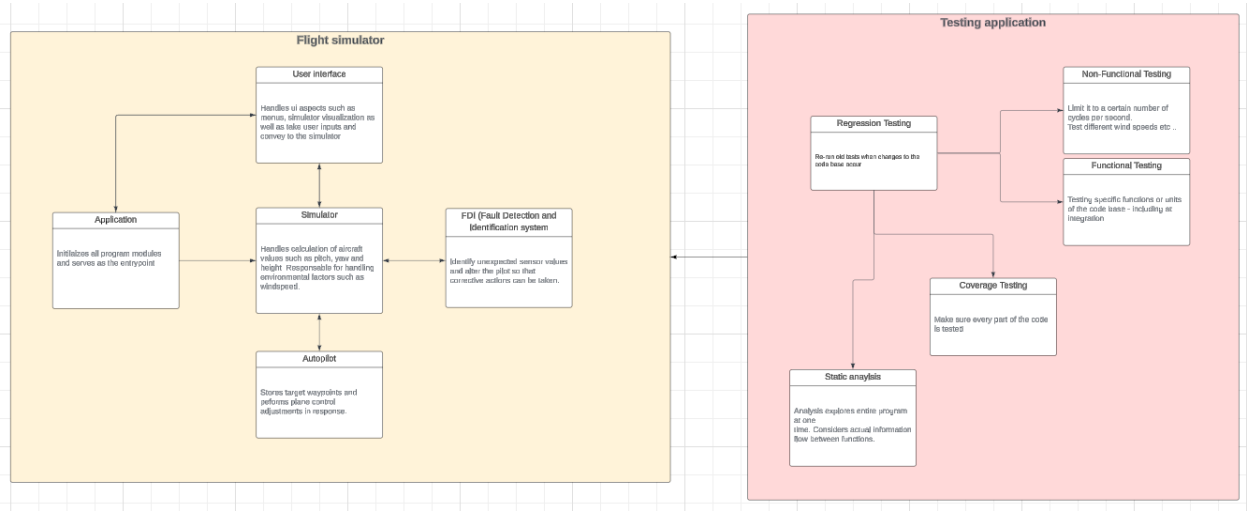
Microkernel architecture:

Module:	Purpose:
Application - Jackie	This module should contain the main method of the application. - Handles initialization of modules as well as program lifecycle - - Does not actually have buttons or much functionality. The application module is sort of like the glue for the entire program. - Choose which aircraft to load. The aircraft should be stored in JSON files and loaded (Flight_details.json) - Also set the start point and destination in the simulation.
Simulator – Nicolas	Handles the simulation of aircraft and calculation of values. Loads information on plane stats, as well as maps "Simulator" class could be the main class for the simulation, and could have references to other classes from within this. The Simulator module could be represented as a module with classes e.g. - Simulator.java - Map.java - Flight.java - Etc... **Map:** For now, assume that the aircraft stays in the air and goes from point to point, and lands. Could have a "World" class and have X and Y for objects (e.g. plan). This could be displayed in the User interface module. **Sensors:** 2oo3 (2 out of 3) agreement with the sensors for redundancy. For simplicity, have 3 sensor classes in a list, e.g. List<Sensor>.

	The values will have a random chance of reporting incorrectly. In simulation intialisation arguments, can add sensor accuracy. - The program will calculate the expected sensor value, e.g. 55. The three sensors will return 55 if the inaccuracy is 0. If the inaccuracy is 0.3 (30%), this may mess up the sensor values.
FDI (Fault Detection and Identification system - Kunal	Observe sensor values, and take actions to alert the pilot if the values are off like using predefined criteria i.e. (Air speed is too low or high, altitude loss) Responsible for sending alerts to the user interface at various urgency levels. Simulates scenarios like sudden loss of thrust, incorrect parameter. The system should be able to perform corrective actions if sensor values are abnormal. For more details, look at the low-level requirements.
Autopilot – Patrick mills	When enabled, performs course corrections in response to waypoints. Contains methods to load and append waypoints. Should be able to be overridden at any point.
User interface (for pilot etc...) - Patrick sawyers	This module will display digital readouts like airspeed, altitude, pitch, roll, yah and engine parameters. Should include warning lights and other notification systems to alert the pilot Source:https://en.wikipedia.org/wiki/Electronic_centralised_aircraft_monitor Levels 1-3 failures, as well as advisory and information messages **Map:** Display the 2D map of the flight plan, which is stored. Allows the user to set a flight plan and waypoints.

Testing application:

Non-Functional Testing	Limit it to a certain number of cycles per second. Test different wind speeds etc... running the simulator with tons of various values. - The actual application itself will run with realistic time delays. The testing program can either run with realistic time or instantly. This is because we don't want to wait forever for a test to run. - We could either run an environment test individually or run many of them at once.
Functional Testing	Testing specific functions or units of the code base - including at integration
Regression Testing	Re-run old tests when changes to the code base occur
Coverage testing	Make sure every part of the code is tested
Static analysis	Intraprocedural. Analysis limited to exploring single method at a time. Assumes all valid information flow between two functions is valid. Intraprocedural. Analysis explores entire program at one time. Considers actual information flow between functions.



https://lucid.app/lucidchart/7dfddfab-8156-4202-a644-9f085002461b/edit?viewport_loc=-1818%2C-782%2C3454%2C1503%2C0_0&invitationId=inv_0dfb13ae-667d-44ce-8622-42447dd61fd2

Hazard Analysis

Hazard	Design Constraint
[Example only] Train starts with door open.	[Example only] Train must not be capable of moving with any door open.
Sensor data not updating within expected frequency	Implement a mechanism to trigger an alert if sensor data does not update within the specified frequency.
Incorrect sensor data leading to incorrect calculations	Implement redundancy in sensor components and require agreement from at least 2 out of 3 redundant components before accepting sensor readings. If all components disagree, display a critical warning to the pilot.
Thrust asymmetry causing yaw and roll	Develop algorithms to detect and correct thrust asymmetry, resulting in proportional yaw and roll adjustments. Display a critical warning to the pilot if thrust asymmetry persists.
Autopilot failure to disengage properly	Ensure that manual override controls are functional even if autopilot fails to disengage. Display the current mode clearly to the pilot.
Exceeding maximum roll limits	Implement a mechanism to automatically correct the roll if it exceeds 25 degrees in either direction. Display a critical warning to the pilot.
Failure to adjust thrust during altitude changes	Implement a mechanism to adjust thrust levels accordingly to maintain the desired altitude. Display a critical warning if thrust adjustment fails.
Inability to add, remove, or modify waypoints	Provide intuitive controls on the user interface to enable users to add, remove, or modify waypoints. Ensure that changes reflect in real-time on the flight plan display. Display a warning if waypoint modifications fail to update the flight plan.