CISC 322/326 Assignment 3: Proposed Enhancement of FlightGear

April 12th, 2024

Group 3

Team Name: Project Flight Forge

Name & Student Numbers:

Lucas Coster (20223016)

Divaydeep Singh (20143859)

Antonio Sousa-Dias (20289507)

Alan Teng (20312270)

Donovan Bisson (20230334)

Zhongqi Xu (20200067)

Table of Contents

1. Abstract

This proposal outlines a plan to enhance FlightGear, a flight simulation software, by introducing a customization module. The enhancement aims to provide users with a more personalized and engaging simulation experience. It discusses the motivation behind the enhancement, explores interactions with existing subsystems, and presents two implementation approaches. Through SAAM analysis, stakeholder concerns are considered, guiding the selection of the preferred implementation. The proposal also addresses potential impacts on maintainability, evolvability, testability, and performance. Additionally, testing strategies and potential risks associated with the enhancement are discussed. Overall, this proposal offers a concise roadmap for enhancing FlightGear, focusing on user satisfaction and technical feasibility.

2. Introduction and Overview

The Flightgear software has been growing in terms of popularity over the past years with the software being used for educational and industrial purposes. Some users just want to use the software to simply experience what it is to pilot an airplane. This report aims to provide a feature that enhances the experience of piloting an airplane. This would be done by allowing the users to decorate the aircraft so there is something to interact with while the user goes through the piloting session. This will allow the user to make a unique aircraft fit for them.

In this report the implementation and integration of this enhancement will be outlined along with potential effects and risks it will have on the existing system. The team analyzed the conceptual architecture of FlightGear as is and explained where the new modules would have to be connected to create a working product. It is believed this module will provide a more enjoyable experience for all users and will benefit the FlightGear software.

3. Proposed Enhancement

The group has proposed an interior and exterior customization module for the FlightGear simulator. This enhancement would give users more customizability within the application creating an aircraft tailored to their personality allowing for more personalization and enjoyment.

3.1. Motivation

The motivation behind including the interior and exterior customization module in FlightGear is rooted in the desire to provide users with a more immersive and personalized simulation experience. By allowing users to add decals, bobbleheads, and other customizations to their aircraft, we aim to empower them to make their virtual flying experience uniquely their own. This module adds a layer of creativity and personal expression, fostering a deeper connection between the user and their aircraft.

Before the inclusion of the interior and exterior customization module, the original FlightGear software lacked a significant avenue for users to personalize their aircraft to suit their preferences and tastes. This limitation hindered the user experience by providing a standardized, one-size-fits-all approach to aircraft simulation. Without the ability to add custom elements, users were unable to inject their personality into their virtual aircraft, resulting in a less immersive and engaging experience. Additionally, the absence of customization options limited the sense of ownership and connection users could feel toward their simulated aircraft, potentially reducing long-term engagement and enjoyment. In essence, the lack of customization options represented a missed opportunity to enhance user satisfaction and deepen their involvement in the FlightGear community.

3.2. Interactions

3.2.1. Aircraft

The Aircraft subsystem contains the model and textures of the exterior of the aircraft. Thus, any decals that are to be placed on the outside model of the aircraft will need to interact with the Aircraft subsystem to determine where decals can be placed and how they are rendered in the simulation.

3.2.2. Cockpit

The Cockpit subsystem defines the interior model of a ship's cockpit. Any decorations placed within the cockpit of the aircraft will need to interact with the Cockpit subsystem to define the space where the decorations can be placed and to properly maintain their location within the cockpit during the simulation.

3.2.3. GUI

Aircraft customization requires an interface by which the user can apply decorations to their aircraft. The GUI subsystem handles user I/O during the simulation, as well as any menus available to the player during runtime. Therefore, any implementation of

customization for aircraft that is done within the simulation (such as through a pop-up menu for selecting decorations accessed via a particular button, and in-flight placement of decorations within the cockpit) would interact with the GUI subsystem for the I/O function.

3.2.4. Sound

In the event any decorative elements available to be placed within the cockpit are accompanied by sound effects (for instance, a bobblehead figurine that makes a springing sound when disturbed), those sound effects should be passed to the Sound subsystem for handling during the simulation.

3.2.5. FDM

While static decorations (such as decals) should not involve the FDM, any non-static decorative elements (such as the bobblehead figurine) that are affected by physics will depend on the FDM to control the simulation of any moving parts in response to forces such as gravity, turbulence and other changes in velocity. However, the flight simulation itself should not be affected by the decorative elements added by the subsystem, so external customization would be limited to texture changes only.

3.2.6. Network

One challenge with allowing customization for user aircraft is how any custom assets are rendered by other FlightGear instances on the same network. The Network subsystem handles the communication of important information across a network, including the Aircraft model and position. Custom decorations will also need to interact with the Network subsystem to allow other users within a network to see them.

4. Implementation

To implement the Customization module the team had to investigate the current state of the software, comparing the existing modules with the proposed enhancement to better understand what changes would need to be made. Following that the team proposed implementation techniques and described how they would be implemented in detail.

4.1. Current State

To encapsulate the current state of the FlightGear program we will circle back to the concrete architecture to recall how the software is implemented. This is relative as the customization enhancement will likely have multiple effects on the current state and we must examine the before and after of the program to acknowledge where issues may arise. Currently, the FlightGear software comprises ten main subsystems, each serving distinct

functions within the simulation environment. These subsystems include Navaids, Sound, FDM (Flight Dynamics Model), Cockpit and Instruments, Aircraft, GUI, Autopilot, AI Model (Artificial Intelligence Model), Air Traffic Control (ATC), and Scenery and Environment.

Upon adding the Customization module into the architecture, it is presumed the GUI, and Cockpits and Instruments modules would be the most affected but multiple other modules including sound, FDM or the Aircraft module could also be affected.

The Cockpit and Instrumentation module serves as the backbone for managing all controls, instrument displays, and visual elements within the aircraft's cockpit environment. Its primary function revolves around handling user interactions with the cockpit interface, ensuring that inputs are accurately interpreted and reflected in the simulation. With the addition of the interior customization module, the Cockpit and Instrumentation subsystem would likely undergo enhancements to accommodate the integration of personalized elements such as decals or bobbleheads. This may involve extending its functionality to support the display and manipulation of custom items within the cockpit environment, ensuring seamless integration with existing cockpit controls and displays.

Similarly, the GUI module in FlightGear currently acts as the conduit for communication between the simulation's backend and the front-end user interface. It facilitates all visual and graphical representations within the simulation, including cockpit displays, scenery, and user input controls. With the introduction of the customization module, the GUI subsystem would play a crucial role in providing the necessary user interface elements for accessing and manipulating customization options. This would involve designing intuitive menus or panels within the GUI to enable users to select and apply custom decals, bobbleheads, or other personalized elements to their aircraft. Additionally, the GUI would handle the visualization of these customizations within the simulation, ensuring that users receive real-time feedback on their modifications.

4.2. Implementation Proposal

To provide the customization functionality that the team aims to enhance some form of a dataset and a decoration module must be implemented and integrated. Outlined below are two implementation techniques that explain how the modules could be integrated into the report, comparing in-simulation customization versus out-of-simulation customization.

In both approaches, modules are included to handle the new decorations required, and the connection between these new modules is explained. The repository architecture style also stays consistent across both implementations. The implementations describe the

conceptual architecture of FlightGear with the new module included and show a diagram of how the architecture may look after implementation. It is assumed that the implementations will have the most effect on the Cockpit, Aircraft, and GUI modules but it is expected that unknown dependencies will arise as the implementations are developed further.

4.3. Implementation #1

Since placing decorations was not considered by the designers at the time when the airplane cockpits were first designed and considering that most of the cockpits were created by community contributors; creating placement of decorations for each cockpit and adding decoration models directly within the game could result in a huge amount of work. Thus, the first implementation will be an in-game editor function that can read models from specific folders and add/adjust the placement of decorations. This gives the user the freedom to place any decorations they like while in-flight.

FlightGear's 3D model consists of three main components: the texture map, XML configuration file, and the 3D model itself. The texture maps and models are usually done by the creators when designing with AC3D or Blender, so developers only need to deal with the XML file used by FlightGear, which determines where a model is placed in the game world and animation. FlightGear reads the aircraft model along with the cockpit model, instruments model, and aircraft's XML file. The code snippet for the instrument in the cockpit is placed within the XML file for the aircraft model usually named "aircraftname-model.xml". In the code snippet as shown below (Figure 4.3.2), <*path*> determines the location of the instrument's 3D model. <*offsets*> specifies the position of this instrument relative to the cockpit of the airplane. Using this feature, developers can think of decoration as an instrument that does not interact with the state of the airplane or the user's input. Therefore, in principle, the decoration editor in this implementation puts the user-selected decoration into the cockpit scene, changes the position of the item according to the user's input, and finally saves the user-selected position and the corresponding path of the model in the aircraft model's XML file.

In this implementation, the architectural style will remain unchanged as the repository style. The conceptual architecture is shown below (figure 4.3.1) The aircraftname-modle.xml acts as the central data structure containing the position and animation of the model. The decoration editor operates on the aircraftname-model.xml file (i.e. central data structure). It is efficient to deal with multiple decorations. FlightGear agrees on the XML model a prior while this function module does not require distribution data.

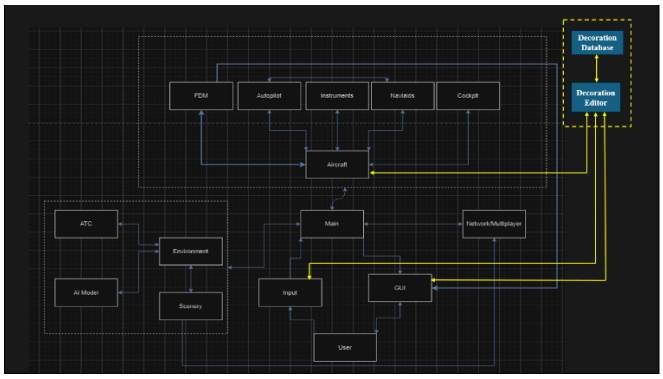

Figure 1: Conceptual Architecture of Implementation #1

Two subsystems will be added:

Decoration Editor: The decoration editor can edit the relative position of the decoration to the aircraft interior.

Decoration Subsystem: The decoration subsystem stores all decoration model information.

Four dependencies will be added:

Decoration Editor <-> Decoration: The Decoration Editor will read the models 'path provided by the Decoration subsystem.

Decoration Editor <-> Aircraft: The Decoration Editor reads the Aircraft's XML file, and cockpit model and saves the decoration's XML code snippet into the aircraft's XML file after the user clicks the save button.

Decoration Editor <-> GUI: The GUI renders the cockpit and the decoration model selected in the Decoration Edit.

Decoration Editor <-> Input: The user adjusts the position of the decoration relative to the cockpit via the input subsystem.

```
<!— ITT Gauge —>
<model>
  <path>Aircraft/fokker50/Models/itt.xml</path>
  <offsets>
    <x-m>-8.9088</x-m>
    <y-m>-0.0811</y-m>
    <z-m>-1.0722</z-m>
    <pitch-deg>-10</pitch-deg>
  </offsets>
</model>
```

Figure 2: User Adjustment Pseudocode [1]

4.4. Implementation #2

Another implementation of aircraft customization is via a placement system that operates in a separate editor engine, rather than within the simulation. Any customizations made to an aircraft will be handled similarly to Implementation 1, with decoration objects stored within a subsystem and pointers to decorations on an aircraft within the Aircraft XML file. The Customization Engine differs from the in-flight editor in that all edits are made on a separate application that can be launched from the FlightGear main menu. This application would feature its renderer in the style of popular model editors such as Blender, allowing the Aircraft models to be repositioned to aid in decoration placement. It adopts a repository-style structure, with its Editor, Renderer, Input, and GUI subsystems linked by the Main subsystem to run the editor separately from the architecture of FlightGear itself (save for any information about Aircraft to generate the models).

The advantage of this approach is twofold. Firstly, the Customization Engine itself can be approached as a third-party repository-style architecture (like FlightSim, the simulation engine for FlightGear instances). This means the decoration suite can be iterated upon to add a variety of useful features to improve usability, such as scaling and rotating objects or mapping decals onto the Aircraft model. Secondly, Decorations as an object-oriented subsystem (like the Aircraft and Environment subsystems) allows for ease of access for users to find and create Decorations for public use.

The addition of a dependency between Decorations and Network/Multiplayer allows communication of customized Aircraft between instances across a network to render and share between users.

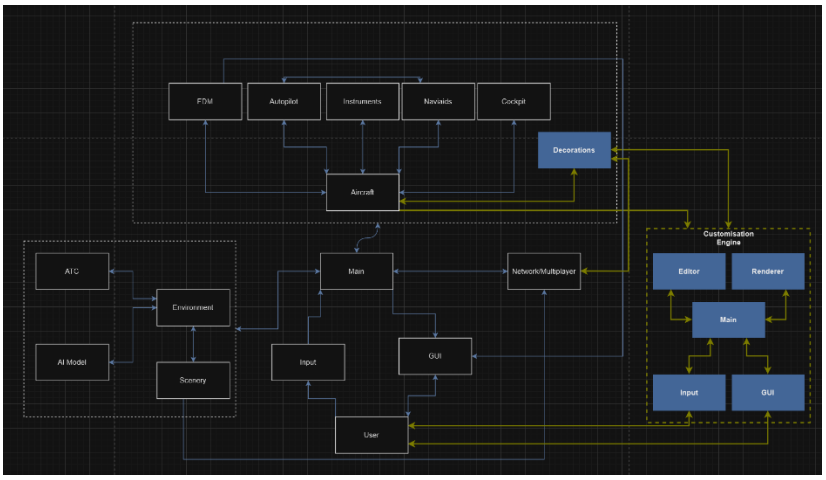

Figure 3: Conceptual Architecture of Implementation #2

5. SAAM Analysis

The following section provides a SAAM architectural analysis between the two implementations being considered for implementing a plane customization and cockpit building module for FlightGear. The team intends to compare major stakeholders, relevant non-functional requirements, and the impacts of implementing the enhancement. The information is then used to determine the most feasible approach for implementing the feature.

5.1. Stakeholder Analysis

The team identified the following stakeholders:

Users: The end users for FlightGear include aviation enthusiasts, pilots, and simulation gamers who use FlightGear for training or entertainment. This group will primarily be concerned with usability, performance, reliability, and compatibility.

Developers: The individuals or teams who work on developing FlightGear and its dependencies, including FDMs, cockpit functionalities, and user interfaces. This group will primarily focus on performance, scalability, interoperability, security, and maintainability.

5.2. Impact on NFRs and Stakeholders

Users

Table 1: Users NFR Impacts

	Implementation 1	**Implementation 2**
NFR		
Usability	By adding an in-game editor, users will be able to modify their aircraft as per their preferences. Since this editor operates within the sysyem, new users can easily adapt to this update.	An external decoration module can increase the learning curve for new users. Additionally, the transfer of data between an additional module may use more resources on the user's host computer.
Performance	System performance may be impacted, especially on lower-end systems due to increased complexity in FlightGear's main loop.	By separating the editing from simulation, performance is optimized at the cost of increased operational complexity.
Reliability	Adding editing code directly in the main loop might impact reliability as errors in the	Since there are no changes to the existing components in the system, the reliability of the system is unaffected.

	editor can affect overall system operation.	Errors in the customization module will not affect system operation.
Compatibility	Since this implementation involves changing the source code of the system, compatibility issues may arise between different FlightGear versions especially during multiplayer sessions.	Since there is no change to the existing codebase, the compatibility of the system is unaffected. The users who want customization can download the external module separately.
Overall Impact	While most users may not notice a significant impact on performance, this change would significantly impact performance and response times. The compatibility issues arising out of this module are also significant. These issues make this implementation unsuitable.	This implementation trades computational complexity for optimized performance and addresses compatibility issues through modularity. This implementation is suitable as it maintains required NFRs by adding a slight computational overhead.

Developers

Table 2: Developers NFR Impacts

	Implementation 1	**Implementation 2**
NFR		
Performance	System performance may be impacted, especially on lower-end systems due to increased complexity in FlightGear's main loop.	By separating the editing from simulation, performance is optimized at the cost of increased operational complexity.
Interoperability	Adding code within the main loop will require additional work to ensure the code is compatible with all other modules and functions.	This implementation has better long-term interoperability as it acts as a standalone module. The developers can ensure compatibility with other modules.
Security	This implementation does not introduce significant new security concerns as it operates within FlightGear's environment. Some security concerns might occur due to users manipulating game files.	This implementation might introduce separate security considerations. However, it is easier to address security in a dedicated environment. Users modifying files in this environment will not significantly impact FlightGear's environment.

Maintainability	Since this implementation is directly integrated within FlightGear, it is difficult to directly maintain without affecting overall code.	This implementation is easier to maintain and improve as it acts as a standalone tool. This module can also adapt new technologies and standards as it evolves with relative ease.
Modifiability	This implementation has high modifiability for users to place and adjust decorations dynamically. This approach is also dependent on FlightGear's architecture for the ease of updating and adding new features.	This implementation has very high modifiability as it can act as an independent tool without affecting gameplay. This approach allows users and developers to add new details such as scaling, rotating, and mapping decals.
Overall Impact	This approach is not ideal due to concerns with interoperability, maintainability and performance. Integrating code directly into existing code reduces performance while introducing new concerns with maintainability.	This approach benefits maintainability, interoperability, and performance. Developers can take additional steps to ensure security while exchanging data.

5.3. Preferred Implementation of Architectural Enhancements

After conducting a SAAM analysis of the two different implementations, the team concluded that implementation #2 is the preferred approach. This decision was made primarily due to concerns about performance, interoperability, and code readability. Since FlightGear implements a modular approach to its codebase, implementation 2 will follow existing guidelines as it introduces an additional module. The enhancement to user experience is identical in both implementations. Finally, the standalone module for decoration can be maintained independently and developers can add new features, details, and standards without affecting FlightGear's existing code.

6. Enhancement Impact

The customizability enhancement will provide a more enjoyable experience for the user but will also have an impact on some high-level aspects of the FlightGear program itself. We will analyze how the change will impact the following non-functional requirements.

6.1. Maintainability

The customization enhancement would likely increase the complexity of the system, particularly within the Cockpit and Instrumentation and GUI modules, as they would need to accommodate new functionalities for managing and displaying customized elements. This increased complexity could potentially make maintenance tasks more challenging, requiring careful documentation, modular design, and adherence to coding standards to ensure ease of troubleshooting and future updates.

6.2. Evolvability

The enhancement could increase the evolvability of the FlightGear system by providing a flexible framework for adding new customization features in the future. By designing the system with extensibility in mind, developers could more easily introduce additional customization options, such as new types of decals or customizable cockpit layouts, without necessitating significant modifications to existing code. This would allow FlightGear to adapt to evolving user preferences and technological advancements over time [3].

6.3. Testability

The introduction of the customization enhancement would necessitate the development of comprehensive testing procedures to ensure the stability and functionality of the system. Test cases would need to cover a wide range of scenarios, including the addition and removal of custom elements, compatibility with different aircraft models, and performance under varying system configurations. Automated testing tools and techniques, such as unit tests, integration tests, and regression tests, would be essential for validating the correctness and robustness of the customization features without compromising the integrity of the FlightGear simulation [5].

6.4. Performance

It could potentially impact the performance of the FlightGear system, particularly in terms of rendering and simulation speed. The addition of custom decals, bobbleheads, and other personalized elements may increase the computational overhead required for rendering the aircraft's interior and exterior models. To mitigate performance bottlenecks, developers would need to optimize rendering algorithms, implement efficient data structures, and leverage hardware acceleration technologies where applicable. Additionally, monitoring, and profiling tools could be employed to identify and address performance issues proactively, ensuring that the customization features do not degrade the overall simulation experience [4].

In summary, while the customization enhancement has the potential to enhance the user experience of FlightGear, it also presents challenges in terms of maintainability, evolvability, testability, and performance. By adopting best practices in software design, development, and testing, developers can effectively address these challenges and ensure that the customization features are seamlessly integrated into the FlightGear system while maintaining its stability, reliability, and performance.

7. Sequence Diagrams

7.1. Use Case #1: Launching a Simulation

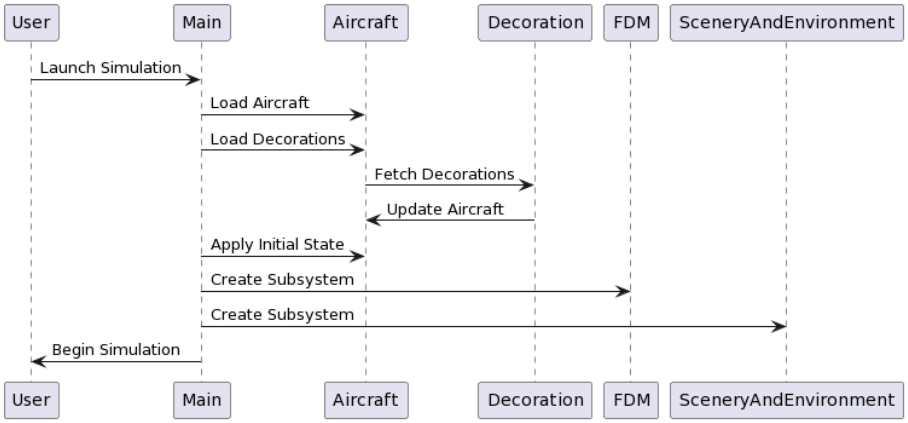

Figure 3: Launching a Simulation Sequence Diagram

Assuming the feature is implemented using the proposed implementation #1, launching a simulation begins when the user prompts the main module to begin a simulation. The aircraft module first loads the originally selected aircraft then updates the previous customizations made to the aircraft, then fetches the model information of the decorations being used from the decoration submodule, where the data is stored. The aircraft is updated with the decorations, then the process continues as in the concrete architecture. Main applies the initial state of the aircraft and creates the FDM and Scenery & Environment subsystems. The simulation then begins.

7.2. Use Case #2: Editing Decorations

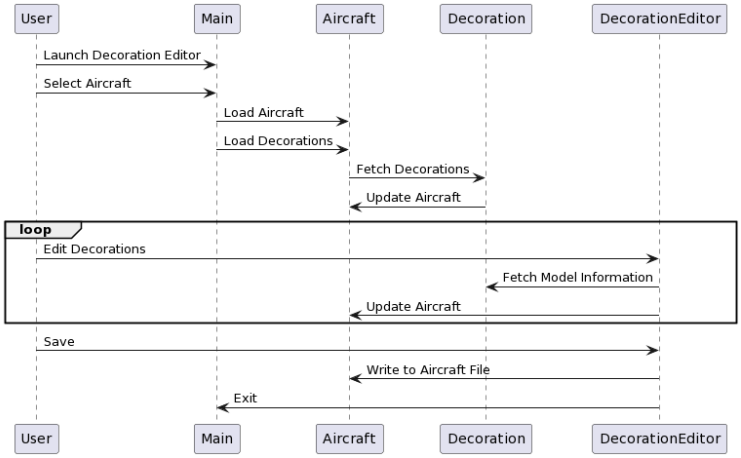

Figure 4: Editing Decorations Sequence Diagram

Assuming the feature is implemented using the proposed implementation #1, editing decorations begins with the user selecting the decoration menu, and then selecting the aircraft model to edit. The aircraft module loads the selected aircraft and then loads any previous decorations made to the aircraft, which are fetched from the decoration submodule. The user interacts and edits the decorations, communicating with the decoration submodule which stores decoration model information. The edited decorations are stored in the aircraft's xml file, to be loaded into a simulation. The user exits the decoration menu and returns to the main menu.

8. Testing

The addition of decoration information attached to the Aircraft subsystem will require adjustments to the existing test code. Reformatting the Aircraft XML document to include information regarding any decorations may require existing integration tests for the Aircraft subsystem and its dependents to be edited and verified for backwards compatibility to account for the change in dimension (i.e. new columns for decoration pointers and location context may incidentally break interactions with other subsystems that rely on the XML file to dictate aircraft functionality). The Network and Aircraft subsystems would also require new tests concerning the Decoration subsystem, to ensure information passed between subsystems is of the proper format and that information relevant to other subsystems is passed on (i.e. physics information on a set of fuzzy dice to/from the FDM for simulation).

As for the within-sim and outside-sim editors posited in Implementation 1 and 2, testing strategies will differ between the two. The addition of new code to the running loop of the

FlightGear simulation would require integration testing for the Decoration Editor subsystem for the Main, Aircraft, Input, and GUI subsystems, and verification that the new running loop code within the Main subsystem does not invalidate existing functionality via adjustments to existing tests. As for the external editor, tests that ensure the Customisation Engine functions properly and that any edits made are of the proper format are necessary, as well as testing the FlightGear main menu can launch the Customisation Engine. That said, given the Customisation Engine is an external application, the impact of changes within the Engine itself will not jeopardize functionality within FlightGear in other areas, so tests present in the within-sim implementation can be excluded.

9. Potential Risk

The customization enhancement in FlightGear introduces several potential risks that need to be carefully addressed to ensure the overall integrity and functionality of the system:

Security: Introducing customization options could potentially create security vulnerabilities if not implemented and managed properly. For example, allowing users to upload custom files or scripts could open the door to malicious code injection or exploitation. It's essential to implement strict validation and sanitization measures to prevent unauthorized access, protect against code injection attacks, and ensure the integrity of user-provided customizations.

Maintainability: The customization enhancement may increase the complexity of the system, making it more challenging to maintain over time. As the number of customization options grows, the codebase could become more convoluted and difficult to manage. To mitigate this risk, developers should adhere to modular design principles, maintain clear documentation, and implement consistent coding standards. Additionally, regular refactoring and code reviews can help ensure that the customization features remain maintainable and scalable as the system evolves [2].

Performance: The addition of customization options could potentially impact the performance of the FlightGear system, particularly in terms of rendering and simulation speed. For example, rendering custom decals or intricate cockpit designs may require additional computational resources, leading to increased system overhead. To address this risk, developers should optimize rendering algorithms, minimize unnecessary computations, and leverage hardware acceleration technologies where applicable. Performance testing should also be conducted to identify and mitigate any performance bottlenecks before deployment [4].

User Experience: Poorly implemented customization options could potentially detract from the overall user experience of FlightGear. For example, overly complex customization interfaces or unintuitive controls may confuse or frustrate users, leading to dissatisfaction with the simulation. To mitigate this risk, developers should prioritize user-centric design principles, conduct usability testing with a diverse range of users, and solicit feedback from the community to iteratively improve the customization features [6].

10. Conclusion and Lessons Learned

Throughout this report, several key lessons have emerged. Firstly, prioritizing user needs and focusing on enhancing user experience is essential in software enhancements like the one described. Secondly, conducting a thorough assessment of feasibility ensures that proposed features can be effectively implemented without compromising the effectiveness or performance of the system. Lastly, stakeholder analysis plays a crucial role in understanding multiple perspectives and aligning enhancements with user expectations and development constraints. These lessons highlight the importance of user-centric design, technical feasibility assessment, and stakeholder engagement in the successful enhancement of software systems.

The proposed enhancement for FlightGear presents a promising opportunity to elevate the user experience within the flight simulation community. Through analysis of stakeholder concerns, consideration of non-functional requirements, and exploration of implementation strategies, this proposal has provided a roadmap for integrating customizable features into the software. The selection of the preferred implementation approach was informed by an understanding of user needs and technical constraints, ensuring a balance between innovation and practicality. Additionally, discussions on testing strategies and potential risks have supported the importance of quality assurance practices in software development. It is imperative to address maintenance challenges and monitor performance impacts to ensure the long-term success of the enhancement. Overall, the enhancement will improve the FlightGear software providing a more enjoyable experience for all users.

Reference

1. FlightGear Wiki. "Howto: Creating 3D Instruments." [Online]. Available: https://wiki.flightgear.org/Howto:Creating_3D_instruments. [Accessed: 11-Apr-2024].
2. J. Smith and L. Johnson, "Ensuring Security in Software Customization: Best Practices and Strategies," in *International Conference on Software Engineering (ICSE)*, 2021.
3. R. Brown and M. Williams, "Maintainability Guidelines for Software Customization Projects," *Journal of Software Engineering Practice*, vol. 23, no. 3, pp. 345-362, 2019.
4. P. Jones and A. Garcia, "Performance Optimization Techniques for Customizable Software Systems," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2020.
5. Q. Chen and S. Lee, "Compatibility Testing Approaches for Customizable Software Applications," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 210-225, 2018.
6. M. Nielsen and K. Smith, "User Experience Design Principles for Customization Interfaces," *International Journal of Human-Computer Interaction*, vol. 31, no. 4, pp. 289-305, 2017.