FlightGear Enhancement Proposal

Group 4, Chicken AI

CISC 322/326
Instructor: Amir Ebrahimi
April 12, 2024

Abbey Cameron - 19aec6@queensu.ca
Akash Singh - 20as166@queensu.ca
Derek Youngman - 20day1@queensu.ca
Marion Anglin - 20mma2@queensu.ca
Shrinidhi Thatahngudi Sampath Krishnan - 21stsk@queensu.ca
Ximing Yu - 20xy@queensu.ca

Abstract

This paper analyzes the hypothetical addition of a “plane crash simulation” to the FlightGear system. Currently, FlightGear lacks a realistic crash simulator, which limits the authenticity of the flight experience it offers. The primary motivation behind introducing realistic crash simulations is to provide users with a better gameplay experience. Adding this feature would not only change existing interactions between components but also possibly add new components to the system.

This paper will discuss the motivations behind this enhancement, analyze various impacts of this feature on the development lifecycle, apply the SEI SAAM analysis method to different implementation proposals, explore some use cases for the new feature, and enumerate the functional and non-functional requirements associated with it.

Finally, the report considers the ramifications of this change in areas such as testability, performance, and the overall maintainability to accommodate this change, ensuring a seamless integration process. Moreover, the report outlines testing plans and emphasizes the importance of meticulous planning and execution. The report concludes with a discussion of the lessons the team learned throughout the process.

Introduction and Overview

FlightGear is a flight simulator that is used for pilot training and entertainment purposes. This simulator provides a practical flying experience through various features like authentic flight controls, accurate world view, and weather, as well as realistic flight aerodynamics. FlightGear is available for a multitude of platforms, including Windows, MacOS & Linux. Not only this, FlightGear also supports a multiplayer feature, making it appealing to those who use FlightGear as a video game for entertainment.

A previous report focused on the conceptual architecture of FlightGear, derived by exploring various books, Wikis, and other research papers to gain a better overview of the software. The next report then focussed on the concrete architecture of FlightGear, discussing the similarities and differences found between the derived conceptual and actual concrete architecture of FlightGear.

This report outlines the hypothetical addition of a new feature of the software which is a “plane crash simulation” and will look at this feature’s effects on the system. This includes an outline of a large number of interactions between features.

The first section of this paper begins with an enhancement proposal. This is a high-level description of the proposed feature. This description does not touch upon the technical side of the software, but rather user experience. This includes its integration within the UI and a few reasons why it improves the final application. As well, this section explains the current state of the software with an investigation of the current scenario that plays out upon a crash landing.

Then, after considering the effects of the enhancement on the maintainability, evolvability, testability, and performance of the system, the impacts on the higher/lower level software and also the way this will impact interactions with other components on the software is considered.

Finally, two specific implementations are presented, with a more detailed description of the architecture. These two implementations are compared relative to the non-functional requirements relevant to the relevant stakeholders. One implementation is chosen in favor of the other.

In doing this report, the group learned of the importance of ensuring adding new features to preexisting software is done with patience, care, and through the use of the right methods.

Proposed FlightGear Enhancement - Aircraft Crash Simulation

The proposed enhancement for FlightGear that will be discussed throughout this paper is the addition of a plane crash simulation. The motivation behind this addition to FlightGear comes from the lack of a realistic crash simulator in the system. Currently in FlightGear, if the user crashes, the plane just stops working and maybe blows up or lights on fire [1]. Under the proposed enhancement, crashes would be more dynamic; the effects of a crash would greatly depend on the nature of the crash. This would include new visual effects of the crash, for example, perhaps the plane is still somewhat functional, and its wheels can carry it some distance on the ground, or perhaps the plane splits in half, etc.

During a real plane crash, various parts of the plane will be damaged, and controls of the aircraft will be affected. This is not yet implemented in FlightGear. As well, FlightGear currently models random engine failure, but no mechanical or instrument failure [1]. This enhancement would include mechanical and instrument failure, both randomly and in correspondence to a crash. While FlightGear was not originally made for simulating plane crashes, this team believes that this addition would create a more realistic and complete experience.

This enhancement would feature more than just the addition of more realistic crash simulations, but upon a crash, the user would be provided with feedback on their flight; the severity of their mistakes, as well as how fatal the crash was. This would allow users to reflect on their flight, analyzing things that could be done differently to change the outcome of their landing. In addition, another feature proposed for this enhancement would include the distinction between landing on land versus water in emergency situations. This will provide users with more ways to land their aircraft realistically, as currently, the water landing system is possible, but not fully realized, as water is not modeled distinctly from non-water [2].

The proposed feature would contain not only enhancements to flight options but also to the animations that take place. The plane would feature animations of elements breaking on the plane, which will affect the navigation and control of the aircraft. This means that depending on the state of the user's aircraft, the landing actions required will vary. In addition, the team proposes that these incidents that cause crashes will be randomized. This means that the user will not know when an incident may occur, but will need to learn to react accordingly to handle the situation and land the aircraft safely, either in water or on land.

Overall, this enhancement will provide users with a more realistic experience. The addition of new ways to land an aircraft, and enhanced graphics to illustrate a crash will create a more authentic experience in FlightGear. Various users have commented on the lack of realistic emergency landings, as well as the crash simulation not being realistic enough [1, 2]. This enhancement will address the requests of users and provide a more lifelike experience for them, which FlightGear aims to provide.

High-Level Conceptual Architecture Effects

The high-level architecture will feature alterations inside its simulation subsystem. The addition of the airplane crash simulation would include changes to the subsystems within the simulation branch. These alterations will take place inside the Flight Control subsystem, the Environment and Scenery subsystem, and the Flight Dynamics Model.

Depending on which approach is chosen for the implementation of the proposed enhancement, it will affect the high-level architecture differently. One option is to use the previously existing subsystems and components and add further connections and enhancements to these components, which will not alter the high-level architecture greatly. The other option would be a new component to help handle the crash simulation. This new component would be added into the Simulation subsystem and would interact with both the Environment and Flight Control subsystems. This approach would alter the high-level

architecture, which is shown in the discussion of each implementation. It would introduce a new component and new dependencies, which will be discussed in more detail later. As later discussed, these two different implementations carry different focuses on the functionality, as well. The alterations that would take place and affect FlightGear's architecture will be spoken of in more detail in the low-level architecture section.

Low-Level Conceptual Architecture Effects

The addition of the plane crash simulation would affect the current architecture of FlightGear in a few ways. These would involve changes to the previously existing environment, aircraft, cockpit, and rendering components. This section outlines the changes within these existing subcomponents.

The changes that would take place in the environment component would feature enhanced details on the environment, including water, nature, and other obstacles like buildings and mountains. This change would allow the user's aircraft to get much closer to these obstacles and allow collision on them, which is currently not in place in FlightGear. This change also touches on the advancement of the rendering system that would need to take place. The rendering system would need to be used to create a more realistic look to the environment, both in general and when a user crashes into a given area.

The aircraft component would need to be adjusted to add a damage model. This model would be used on different aircraft features (wings, engines, etc.), and consider how the damage taken affects the specific area, the flight controls, and the performance of the aircraft. This feature would allow realistic damage to the aircraft, which is not currently included in FlightGear. It would include ways for different elements on the aircraft to break apart, both in the sky and when interacting with different features in the environment. As well, some parts of the aircraft can be modeled to randomly fail.

The cockpit and instruments are the final elements that would be affected in the low-level architecture by the addition of the plane crash simulation. This would work with the newly adapted aircraft features to create a more realistic experience. For example, as a component of the aircraft breaks mid-flight, the player would need to adapt how they are controlling the plane through the cockpit to continue flying. This means not all elements of the plane may be available to use anymore, and the user will need to adapt their landing strategies in emergencies. As well, some instruments can be modeled to randomly fail.

A large component that would be affected by this addition of the crash simulation would be the Flight Dynamics Model (FDM). This component calculates the physical forces that are acting on the aircraft at any time and controlling how it flies, including thrust and lift. With this, the FDM calculations and controls will be affected by things like the environment and weather collisions now. Any external force that will interact with the aircraft to cause a crash will be sent through the FDM for a recalculation to alter the flying of the plane. This will alter how the user can instantiate things like lifting the plane up or accelerating.

Finally, the last component that would be adjusted is the rendering system. This element is overarching and will oversee and affect the previously mentioned components. It will enhance the graphics in the simulation and will allow the game to appear more realistic and therefore further enhance the experience the user gets from using FlightGear.

For reference, *Figure 1* shows the previously derived concrete architecture, excluding the central property tree and the main loop. A "new component" is one that was not present in the originally derived conceptual architecture.

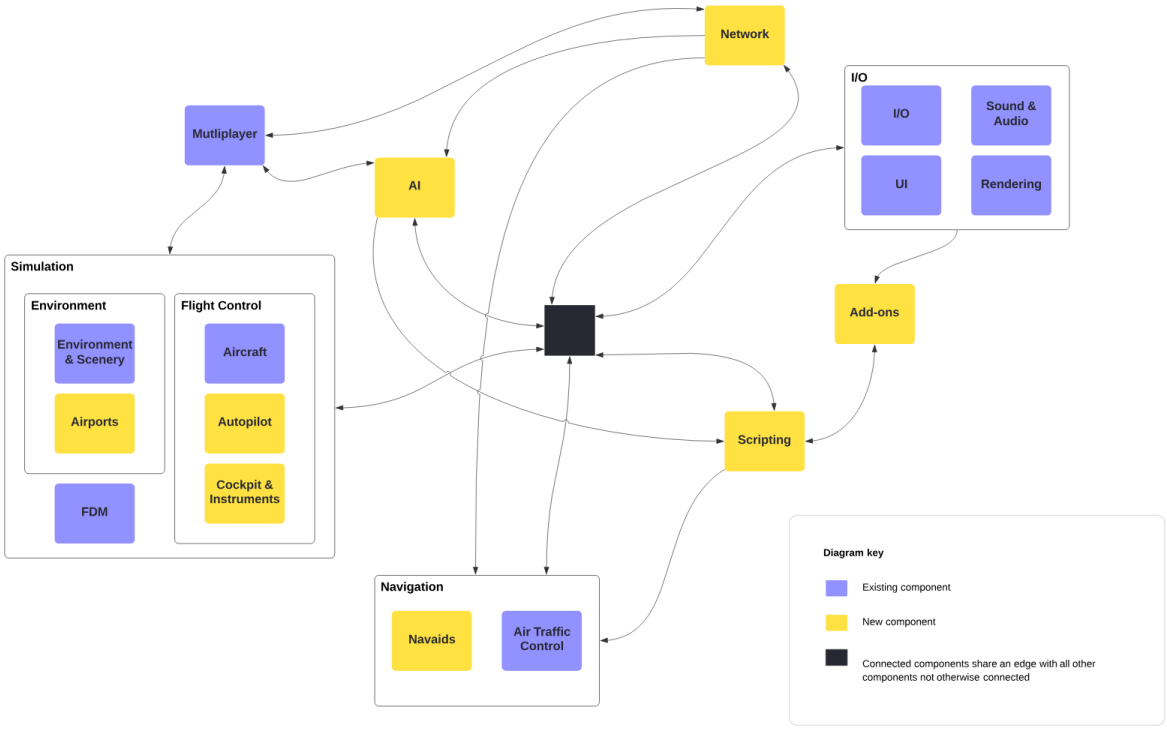

Figure 1: Concrete Architecture of FlightGear's Outer Components

Directories/Files Affected By Enhancement

According to the changes discussed in the low-level architecture, these are files and directories that would certainly require changes.

- **flightgear/src/Aircraft/AircraftPerformance.cxx:** Plane performance changes based on the current state of the plane's health.
- **flightgear/src/Model/:** Add models for planes in various states of damage.
- **flightgear/src/Cockpit/NavDisplay.cxx**: The cockpit simulates danger and warnings to the pilot.
- **flightgear/src/Environment/**: As discussed in the low-level architecture, the environment component requires changes**.**
- **flightgear/src/Instrumentation/**: Add functionality to deal with adapted aircraft handling to account for broken instrumentation.
- **flightgear/src/FDM/**: As discussed in the low-level architecture, the FDM component requires changes**.**
- **flightgear/src/viewer/**: As discussed in the low-level architecture, the rendering (viewer) component requires changes**.**

Testing Plan

Here, a methodology for testing the feature, including its interactions with other features is outlined.

It can be assumed that the developers of FlightGear have utilized an iterative model for FlightGear's development. The Agile method of development is a flexible and iterative approach to software development that emphasizes collaboration, adaptability, and client satisfaction [6]. This seems

like the likeliest methodology for an open-source project to adapt. Unlike the iterative waterfall model, which follows a sequential process, Agile divides the project into small increments, allowing for incremental development and frequent reassessment of priorities.

Within the iterative Agile model, introducing a new feature like crash simulation is integrated into the development process as part of iterative cycles. Testing the impact of the interactions between this enhanced product feature with the rest of the functionality will be done through various testing methods, managed as part of sprint activities.

Functional Testing

A big aspect of functional testing in Agile development involves unit testing, which encompasses both black-box and white-box methodologies. This testing aims to ensure the proper addition of new features throughout the development lifecycle and verifies that all interactions with other features occur as intended. *Black box testing* assesses the external behavior of the feature, while *white box testing* examines its internal functions.

Black Box Test (Output Coverage)

Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input and observes the output generated by the system [7]**.** To evaluate the effectiveness of the plane crash simulation feature, we test a simulation within FlightGear and compare the simulated crash outcome with the expected behavior. The completion criteria is that the simulated crash accurately reflects real-world physics and aircraft behavior.

Black box testing with other features is done by verifying that the collision detection system accurately detects collisions with environmental obstacles such as buildings, mountains, and water bodies. Other architectural features that the enhancement needs to be tested with would be to ensure that the rendering system effectively renders realistic environments and aircraft damage effects without causing any visual glitches or performance issues. Finally, simulating a crash to ensure aircraft handling and instrumentation performance is realistically impacted by the proposed enhancement.

White Box Test (Decision Mutation)

White box testing is a method of software testing that checks the internal structures or workings of an application [8]**.** A white box test for the crash simulation feature involves code coverage analysis. Verifying all relevant portions of the crash simulation algorithm are executed during testing will ensure comprehensive testing of the underlying logic.

White box tests to ensure that the other features of FlightGear are not broken by the proposed enhancement would be to inspect the codebase to verify that the environment, aircraft, cockpit, and rendering components are appropriately modified to accommodate the plane crash simulation feature. Additionally, testing the implementation of the damage model within the aircraft component to ensure accurate simulation of damage effects on various aircraft parts and flight controls.

During implementation, the development team would focus on utilizing unit tests to assess the functionality of the new feature. Additionally, regression testing encompasses the rerunning of both functional and non-functional tests derived from the original specifications and requirements.

Effects of the Enhancement on Maintainability, Evolvability, Testability, and Performance

There is minimal effect of the enhancement on maintainability. Because of FlightGear's open-source nature, adding this enhancement would likely entail new developers who are personally interested in the change helping to implement, so these new distinct developers could maintain it. So, as long as the feature is implemented somewhat independently of the rest of the system, which it can be since it is more of an add-on, there should be very little effect on the overall maintainability of the system. The new feature itself could however be difficult to maintain if the possibly small number of developers dedicated to this new feature lose interest. While there has been some interest expressed for this feature [1, 3], it would still not be a core part of FlightGear, so it is possible that over time developers lose interest in this feature, resulting in this feature being poorly maintained. This is not anticipated, this should be a great feature, but it is possible.

There would be a marginal effect on evolvability. FlightGear is already very complex and interconnected, adding new functionality and a small number of new interactions won't have any huge effect on the evolvability of the already large system. However, there is some limit on how much FlightGear could easily add without having to re-engineer the entire architecture. At some point, if there are too many interactions between components, the little sense of coherent architecture that FlightGear carries would be completely gone, and it might be difficult to make major changes to the system.

The enhancement would have some effect on testability. All of the new features paired together present a large number of new situations to test. For instance, with all the different possible parts of the plane that could fail, there are many possible combinations of parts that could all fail at the same time, presenting many new situations. It would be difficult to provide exhaustive testing for all of the new and different possible scenarios. Thankfully, FlightGear is open-source, so there are lots of people who could find new bugs over time and address them.

There are some possible effects of the enhancement on performance. Modeling crashes, graphically, in particular, can be computationally expensive. The performance might take a hit while crashes are simulated. As long as it's not too bad, it's not a big deal though, when a crash is not happening there will be no impact on performance. As later discussed, in one possible implementation, the graphics could be the focus. In this case, an entire new component could be justified which can be engineered to make sure that the performance does not take a significant hit.

Use Case #1: The Pilot Experiences a Mechanical Failure but Manages to Stabilize the Aircraft

In the first use case, the simulation begins with the aircraft in a normal flight configuration. Then suddenly, a mechanical failure occurs. The pilot has to assess the situation quickly and take appropriate actions to regain control of the aircraft.

During this process, the cockpit & instruments subsystem first displays critical indicators related to the failure, such as oil pressure change, engine RPM, and fuel flow, via the UI subsystem to allow the pilot to diagnose the problem. Then the user implements correct actions by sending mouse or keyboard inputs to the I/O subsystem, which publishes the input changes to the property tree subsystem. The FDM subsystem reads the property changes, does calculations, and accurately simulates the effects of the mechanical failure and the user's actions on the aircraft, including changes in thrust, altitude, and aircraft speed. Any changes in the state of the aircraft are published to the property tree subsystem in real-time. This process continues until the user manages to stabilize the aircraft. Finally, the cockpit & instruments subsystem reads the latest aircraft states and updates the indicators to reflect the restored functionalities.

The pilot successfully averts a potential crash scenario. *Figure 2* presents the sequence diagram of the use case #1.

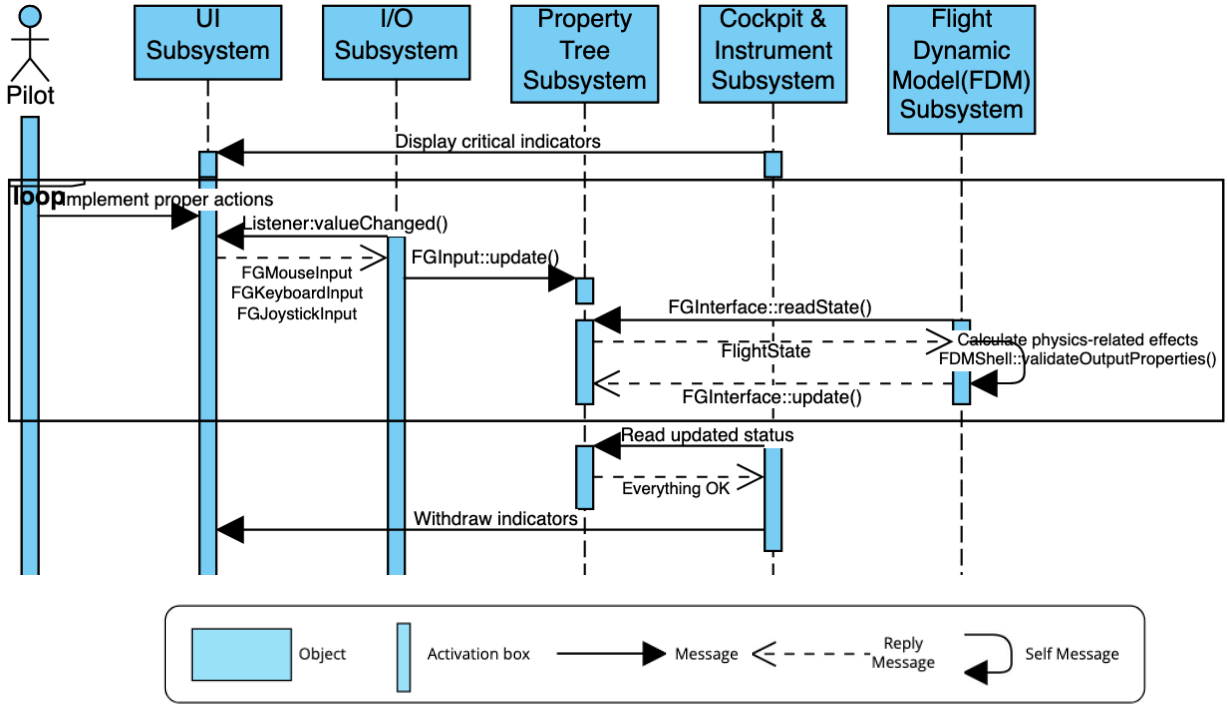

Figure 2: Use Case #1 Sequence Diagram

Use Case #2: The Aircraft Suffers a Catastrophic Failure and the Pilot Has to Make an Emergency Landing on Water

In the second use case, the aircraft experiences severe damage that requires an immediate emergency landing on water. The pilot needs to quickly assess the situation and initiate a ditching procedure.

The pilot adjusts the aircraft's configuration for ditching via the UI subsystem, such as retracting landing gear and configuring flaps for a controlled descent. User inputs are processed by the I/O subsystem and then sent to the property tree subsystem for access by other subsystems. The FDM subsystem simulates the behavior of the aircraft as it descends toward the water surface according to the parameter changes read from the property tree subsystem. The simulation accounts for factors like airspeed, angle of descent, and wind conditions. As the aircraft descents, the environment subsystem needs to continuously read and update the environment parameters in the property tree. The rendering subsystem listens to any environmental parameter changes in the property tree subsystem and renders the updated visuals of surroundings, which are subsequently delivered to the UI. Upon contact with the water, the FDM again calculates the physics aspect of the aircraft's interaction with the water and the environment, and rendering subsystems work together in the way described above to produce detailed and realistic visual effects, including splashing, spray, and aircraft's deformation upon impact.

The interactions between the property tree and FDM, environment, and rendering subsystems happen in a loop during the whole process to deliver a seamless and realistic simulation. *Figure 3* presents the sequence diagram of the use case #2.

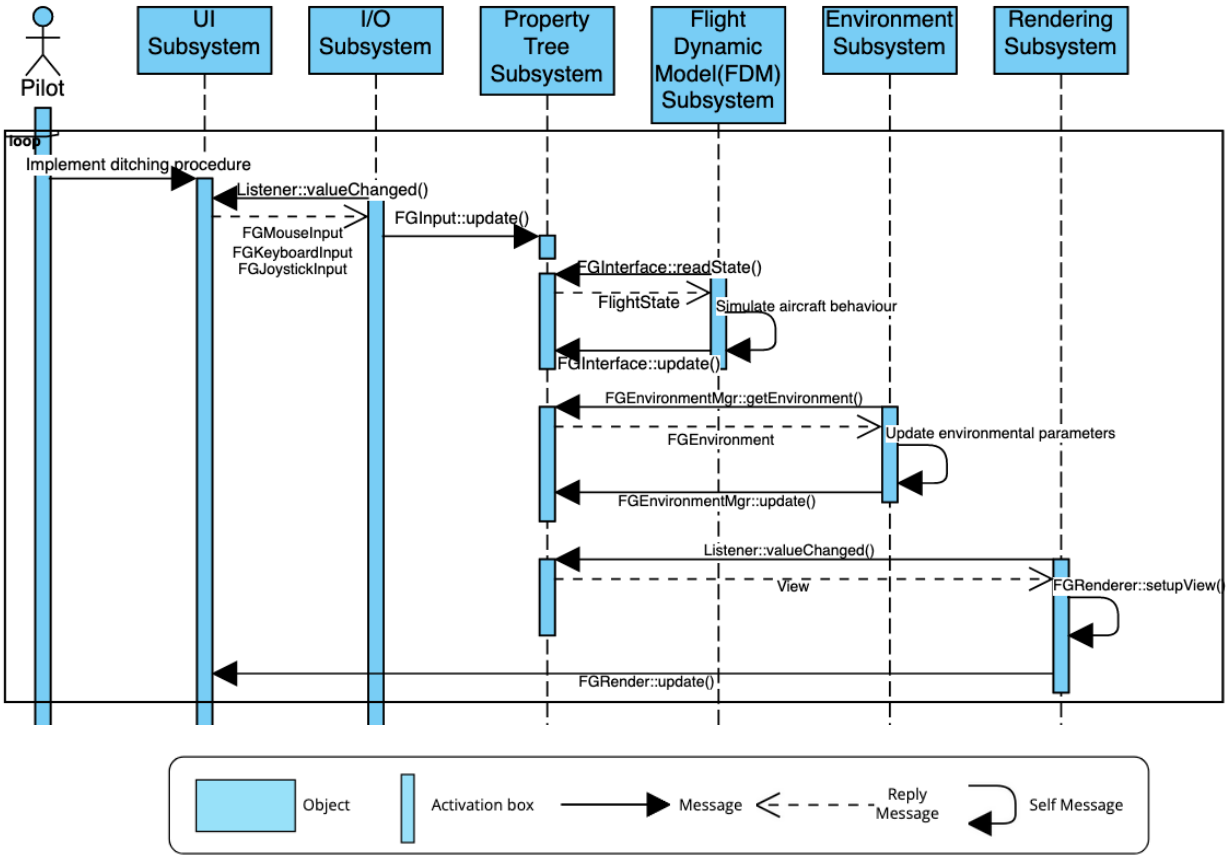

Figure 3: Use Case #2 Sequence Diagram

Implementation Approach 1 – Connected Approach

This team's proposed enhancement aims to refine the behavior of aircraft components under stress and critical situations, such as impacts of structural failures, within the FlightGear simulator. The first implementation approach, the connected approach, does this by increasing and enhancing connections between already existing components. *Figure 4* shows the dependencies required to implement the enhancement in this implementation. The components and dependencies are those relevant to this enhancement only. Other components and dependencies are omitted. Refer to *Figure 1* for a more complete picture. Many of these dependencies already exist in the system for the implementation of other features, but even a repeated dependency between components still adds more dependencies in the system at the subdirectory, file, class, or method level.

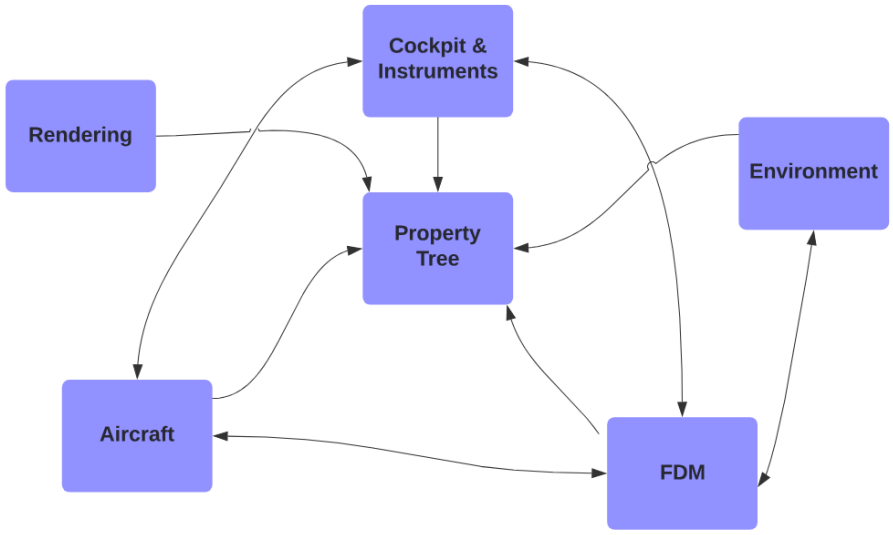

Figure 4: Dependencies Required in Implementation #1 of the Enhancement

In this implementation, the already existing architecture styles for information sharing are maintained. Namely, the repository style and object-oriented styles. The repository style is facilitated by the property tree as the central source of information, while the other components also communicate with each other directly by the use of an object-oriented style, as described in the concrete architecture report.

The Flight Dynamics Model (FDM) plays a central role in updating the status and position of the aircraft within the property tree, which acts as a centralized repository of simulation data. This integration allows various components, including the cockpit instruments and crash rendering system, to access real-time aircraft-specific properties. For example, changes in altitude, airspeed, and engine parameters influenced by the FDM directly impact the behavior of the aircraft as well as the rendering of collision dynamics by the crash feature, enhancing the overall realism and consequence of flight operations.

By enhancing flight physics and dynamics, there will be increased interactions with various other components of the simulator, including the aircraft model, user interface, and sound and visual effects interactions. These interactions will require additional development efforts to ensure that the updated flight dynamics accurately influence and are reflected in the behavior and presentation of these components. Examples of specific interactions may include real-time updates to cockpit instruments based on refined flight dynamics, dynamic adjustments to visual and audio effects based on aircraft behavior, and improved collision rendering and response in scenarios involving impacts of structural failures. This approach will need a comprehensive architectural design to manage the increased complexity and integration of these enhanced flight physics and control dynamics into the existing simulator framework.

The Flight Dynamics Model (FDM) continuously updates critical aircraft-related properties within the property tree, such as position and velocity. These updates are vital for the accurate functioning of cockpit instruments, ensuring that pilots receive real-time, reliable information about the aircraft's behavior and state. The FDM's influence on cockpit instruments, including potential crash indicators or warnings, contributes to a responsive and immersive cockpit experience.

Lastly, the property tree serves as a centralized data repository in FlightGear, facilitating seamless communication among simulator components, including crash rendering, rendering engine, cockpit instruments, and the Flight Dynamics Model (FDM). Each component reads and writes specific properties within the property tree, enabling consistent updates and synchronization across different simulation aspects. This standardized data exchange mechanism ensures effective inter-component interactions, enhancing the overall realism and integrity of the simulator, especially in scenarios involving aircraft crashes and their visual representation.

Implementation Approach 2 – Component Approach

This is the second of two possible more specific implementations of this enhancement. This "component" approach prioritizes the enhancement of crash graphics within the FlightGear simulator, aiming to deliver a more immersive and visually impactful experience during crash scenarios. This approach emphasizes the graphical representation of aircraft impact and destruction while simplifying the underlying dynamics and physical interactions. The implementation introduces a dedicated component, namely the crash rendering component, that processes basic crash-related information and utilizes pipe and filter processing to produce extraordinary graphics depicting crash scenarios. By adopting a modular architecture based on pipe and filter processing, the simulator can efficiently handle crash-related data and generate visually striking effects that enhance immersion and cinematic appeal. The pipe and filter architecture is used here to emphasize the processing speed of the graphics. Given information about the crash scenario, the new component can quickly generate updated graphics, which is computationally expensive, through the pipe and filter architecture. During the processing, this component need not interact with anything else until a new request is generated. The rendering system is still used for more basic graphics which are required more frequently. The focus here is on showcasing the visual effects of crash impact and partial destruction of the aircraft, with less emphasis on simulating the direct impact of these events on actual flight dynamics. This strategy highlights the importance of delivering compelling visual experiences that captivate users and elevate the overall realism and excitement of crash simulations within the FlightGear environment.

Equivalent to the first implementation, the primary architectural styles are maintained for the interaction between components.

The crash rendering system in FlightGear interacts closely with both the environment and the flight control subsystems. By accessing FDM data from the property tree, the crash rendering system accurately simulates collision dynamics and damage effects based on the aircraft's speed, angle of impact, and structural integrity. This realistic rendering, integrated with the flight control and environment data, enhances the visual feedback of crash scenarios, providing pilots with valuable insights into the outcome of their flight maneuvers. All of the dependencies present in this implementation are also present in the previous implementation, however, this implementation lacks some of the dependencies of the previous implementation. This is because this implementation does not focus as much on creating dynamic and realistic crash simulations. The environment component no longer needs to reflect changes relevant to this feature in the property tree, as the environment e.g. water, wind will no longer dynamically change in response to the crashing plane. The cockpit & instruments component also will not depend on the FDM or share any dependency with the aircraft component. In this implementation some of the instruments could still fail, which would affect flight control, however, the functionality of the instruments will not directly depend on the state of the plane, as they would in the first implementation. *Figure 5* shows the dependencies required to implement the enhancement in this implementation. The components and

dependencies are those relevant to this enhancement only. Other components and dependencies are omitted. Refer to *Figure 1* for a more complete picture. Many of these dependencies already exist in the system for the implementation of other features, but even a repeated dependency between components still adds more dependencies in the system at the subdirectory, file, class, or method level.

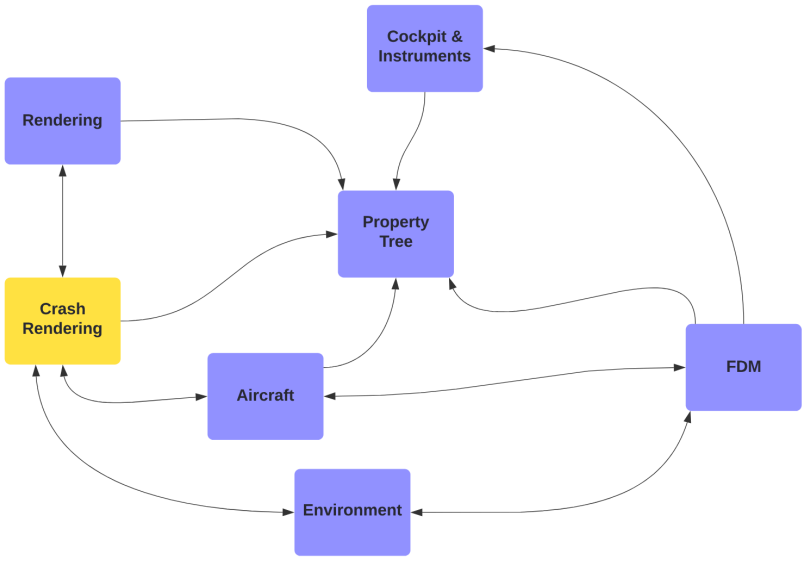

Figure 5: Dependencies Required in Implementation #2 of the Enhancement

SEI SAAM Analysis

There are three major identified stakeholders regarding the proposed enhancement. The first two are different types of users: gamers and pilots. The third one is FlightGear developers.

Gamers would appreciate realism but it is not necessarily the primary concern. Instead, gamers would likely most appreciate compelling graphics and new ways to play the game. Adding new crash scenarios gives gamers more things to explore in-game, apart from the obvious new scenarios, they could even purposely crash the plane in an attempt to get the biggest or coolest crash. Therefore, the biggest NFR for gamers would be performance. If the crash scenarios caused e.g. low frame rates and response times, the user would not be satisfied with the added visual effects of crashing the plane.

Pilots would primarily care for accuracy and realism. If the new crash simulation effects let the pilots realistically explore new situations that they have never been able to practice before, this could add a lot of value to pilot training.

Developers primarily care for maintainability and testability. They don't want their work to be too hard.

Table 1 presents all of the relevant NFRs associated with each stakeholder. *Table 2* compares each implementation relative to relevant NFRs.

Stakeholders	**Primary NFRs**
Gamers	**Performance:** Gamers would want the crash feature to be fast and responsive with

	satisfactory visual effects. **Accuracy/Realism:** Secondary concern, some gamers appreciate realism.
Pilot	**Accuracy/Realism:** The feature needs to be as accurate as possible to be useful and realistic to the pilot. **Modifiability:** Pilots would like to be able to modify the behavior of the aircraft depending on the type of plane they are flying or the scenario they would like to practice. **Performance:** Pilots need their inputs to the system to be responded to quickly to maintain a realistic experience.
FlightGear Developers	**Maintainability:** Developers want the software to be as easy to maintain as possible. **Testability:** Ease of testing is positive for developers, to produce bug-free code. **Evolvability:** Developers would like to maintain the ability to add new features.

Table 1: All relevant NFRs associated with each stakeholder

NFR	Connected Approach	Component Approach
Performance	Possibly lower in terms of graphics, but reasonable in terms of flight control.	High-speed rendering.
Accuracy/Realism	High. The constant interaction with all of the already realistic components of FlightGear can provide a realistic crash experience.	Lower. In this implementation, the "wow factor" of the graphics is prioritized over realistic crash simulation.
Maintainability	Little change. Since FlightGear is open-source, developers who are interested in this feature can freely focus on this feature while other developers focus on other features.	Little change. Since FlightGear is open-source, developers who are interested in this feature can freely focus on this feature while other developers focus on other features.
Testability	Lower. A large amount of new interactions and scenarios present many things to test.	Higher. Graphics present fewer new complex interactions and scenarios compared to dynamic simulation.
Evolvability	Slightly lower. More interactions hinder evolvability in the long run.	Slightly higher. Fewer interactions encourage more evolution.

Table 2: Comparison of each NFR paired with each potential implementation.

SEI SAAM Analysis Discussion and Conclusion

Given the advantages and disadvantages of the implementations relative to each of the relevant NFRs, it seems that the component implementation is better. The first implementation is more suited to pilots, while the second implementation is more suited to gamers and developers. The only clear drawback of the second possible implementation is accuracy/realism. FlightGear is known to be a realistic flight simulator [4], however, the addition of this feature in the second implementation does not take away from the realism of the other features in FlightGear, only the realism of the new feature. Any newly

provided crash simulation capability is still more realistic than the complete lack of crash simulation present in FlightGear right now.

While FlightGear is used for pilot training [4], the complaints of a lack of crash simulation found online do not come from pilots, they come from other users [2, 3]. Also, while pilots could benefit from crash simulation, it is not the primary purpose of the flight simulator, and it would be difficult to provide a completely realistic crash simulation. Therefore, the addition of crash simulation with the goal of attracting gamers or casual players is more worthwhile, especially given the benefits to the developers in this implementation.

In the context of FlightGear, the developers are also one of the most important stakeholders. If any developer is not inclined to do their job, they generally don't have to. Many developers of FlightGear are volunteers. Given the demand from non-pilots, and the benefits of the 2nd implementation to maintainability, testability, and evolvability, the 2nd implementation provides the most motivation to prospective developers to continue to maintain and evolve FlightGear.

Overall, the 2nd implementation will bring the greatest benefit to the largest number of stakeholders.

Risks Associated with the Enhancement

The addition of the crash simulation could impact the performance and stability of FlightGear. Increased computational demands and potential bugs could affect the overall usability of the simulator.

Implementing a crash simulation requires a significant amount of development effort. This could affect all the resources and time devoted to other important aspects of FlightGear. With how complex FlightGear tends to be, integrating crash simulation features seamlessly into the already existing code framework and user interface would present technical issues. The framework for the crash simulation should be compatible across different platforms, and hardware configurations, essential for a smooth user experience

The crash simulation comes with new complexities in terms of testing and quality assurance. Testing is necessary to ensure that the crash simulation behaves realistically and reliably using various scenarios. Testing should cover various edge cases, different aircraft types, and interactions with other features. With the addition of new features, maintaining detailed documentation becomes crucial. The framework should be well-documented to facilitate knowledge transfer among the developers and ensure that future contributors to FlightGear can understand and modify the feature effectively if needed. The code for the crash simulation should be able to evolve easily and extend if need be. When new features/updates are added, the crash simulation should seamlessly integrate with existing systems and have to be adaptable to accommodate future changes to FlightGear. Failure to design accordingly could lead to rigid code that could hinder future development efforts. Scalability issues may arise when simulating crashes with large aircraft or complex environments and meeting real-time performance requirements for dynamic crash simulations might be hard.

Conclusion

From reading through FlightGear forums, the team was able to determine that users felt there was a lack of realistic emergency and crash simulation within FlightGear. Therefore, the team proposed a plane crash simulation feature enhancement that would improve FlightGear's emergency and crash simulation to improve users' overall experience within FlightGear.

The team came up with two possible implementation approaches to realizing the proposed enhancement, namely a connected approach that would increase interaction between already existing components and a component approach where a new crash simulation component would be created to handle crash simulation. Neither approach would alter FlightGear's higher-level architecture, nor would they alter the higher-level repository and object-oriented architectural style. However, both approaches would require lower-level changes in interactions and dependencies of various components within the simulation subsystem.

Following a thorough SEI SAAM analysis, the team determined that the component approach would work best within FlightGear's system and would be beneficial to the most number of FlightGear stakeholders and fit their non-functional requirements the best. Of course, there are risks to both the component and connected approach, but overall, none of these risks are bad enough to not implement the team's proposed enhancement.

Lessons Learned

While working on this assignment, we learned that creative processes can't always be performed from scratch. We found it quite difficult to come up with a proposed enhancement on our own. Looking into additional information for this, namely FlightGear's user forums with various comments from unsatisfied users, was very beneficial in helping us come up with a proposed enhancement. Moreover, we learned that it's almost impossible to find a solution that satisfies all stakeholders and their non-functional requirements. Our solution to which approach to use only fully satisfies our gamer and developer stakeholders. We learned that prioritization and compromises are necessary to achieve results in large-scale systems like FlightGear. Overall, we learned many things about how to propose system enhancements in the real world and will be able to utilize the knowledge we acquired throughout this assignment in our future work experience.

Naming Conventions

- FDM - Flight Dynamics Model
- NFR - Non-Functional Requirement

References

[1] Crash is Not Realistic Enough - FlightGear Forum:
https://forum.flightgear.org/viewtopic.php?f=49&t=16330
[2] Emergency Landing in Water - FlightGear Forum:
https://forum.flightgear.org/viewtopic.php?f=18&t=10290
[3] Crash Simulation - FlightGear Forum: https://forum.flightgear.org/viewtopic.php?f=4&t=30386
[4] FlightGear - Wikipedia: https://en.wikipedia.org/wiki/FlightGear
[5] FlightGear - Source Code: https://github.com/FlightGear/flightgear
[6] Agile: https://www.atlassian.com/agile
[7] Black Box Testing: https://www.imperva.com/learn/application-security/black-box-testing/
[8] White Box Testing: https://www.geeksforgeeks.org/software-engineering-white-box-testing/