SpaceX Rocket Simulator Design and Report

Seal Team 7

COS 214 Group Project

Lindinkosi Mzwethu Kunene - u17102210
Paulus Wilhelmus Mouton - u18069704
Oluwatokesi Daniel Babalola - u18041494
Timo van der Merwe - u1904820
Andrea Blignaut - u19130938
David Martin Roodt - u19264047
Devank Kyle Harripersad - u20440562

Introduction

There are several aspects to a single SpaceX rocket launch: manufacturing rocket parts, storing said parts, assembling rocket parts into viable rockets, “static fire” launch tests and real launches, calculating the total cost of a launch once it has happened, and storing reusable rocket parts back in storage.
This program is meant to simulate those tasks. It does so by using design patterns to implement a complex system.

Links

- Link to Google Doc version of this document: https://tinyurl.com/214Report
- Link to Git repository: https://github.com/andreabeatrice/COS214_Project

Functional Requirements

1 Engineering Requirements

1.1 Component Storage

- The system must be able to add a new component to storage
- The system must be able to return used components to storage after launch
- The system must be able to retrieve parts from storage
- Constraints:
    - If the launch was unsuccessful, all parts are lost and none are returned to storage

1.2 Construction

1. The system must allow a user to input a mass for the cargo that will be launched by the rocket (in kilograms).
2. The system must allow a user to specify the type of cargo that will be launched by the rocket.
    - Constraint 1: The cargo must be of type astronauts, supplies, satellites, or a combination thereof
3. The system must be able to display all viable combinations of rockets (where the rocket is either a Falcon Rocket, a Falcon Rocket with a Dragon Spacecraft attached, or a Falcon Rocket with Starlink satellites attached).
    - The system should be able to model a Falcon 9 first stage with a single Falcon 9 core and 9 Merlin engines.
    - The system should be able to model a Falcon 9 second stage with a single Vacuum Merlin Engine.
    - The system should be able to model a Falcon Heavy first stage with three Falcon 9 cores and a total of 27 Merlin engines.
    - The system should be able to model a Falcon Heavy second stage with a single Vacuum Merlin Engine.
    - The system should be able to model the Crew Dragon Spacecraft to send and safely return Humans and Cargo to the International Space Station.

    - The system should be able to model the Dragon Spacecraft to send only cargo to the International Space Station.
    - The system should be able to model a launch of the Starlink satellites.
4. The system must allow a user to specify the type of orbit that the rocket will be launched into.
    - Constraint 1: The orbit must be of type **Lower Earth Orbit (LEO)** or **Geostationary Transfer Orbit (GTO)**
    - Constraint 2: If the orbit type is **GTO** and cargo mass is>8.3 tonnes, a Falcon Heavy Rocket will be required.
    - Constraint 3: If orbit type is **LEO** and cargo mass is>22 tonnes, a Falcon Heavy Rocket will be required.
5. The system must be able to remove the selected components from storage and add them to a new Rocket.
6. The system must be able to repeat this process.

2 Logistics Requirements

2.1 Launch Batch

1. The system must be able to receive a rocket/a list of rockets and add it to the launch batch
2. The system must allow a user to choose the type of launch.
    2.1. Constraint 1: The launch must be of type real or simulation.

2.2 Launch Cost

1. The system must be able to calculate the cost of the launch and return that calculation to the user.
    1.1. The launch cost is encompassed by:
        1.1.1. The damage done to the components.
        1.1.2. The cost of preparing the launchpad.
        1.1.3. The cost of the fuel used.

3 Launch Simulation Requirements

3.1 Test Mode Simulated Launch

1. The system should allow a test mode simulation to be paused, changed and resumed.
    1.1. The system should allow a user to add a component to the rocket.
    1.2. The system should allow a user to remove a component from the rocket.
    1.3. The system should allow a user to change a component of the rocket.
2. The system must be able to return an appropriate message on a failed launch simulation.
3. The system must be able to return an appropriate message on a successful launch simulation.
4. In the case of a launch with Starlink satellites, the system must be able to test communication between satellites.

3.2 Real Launch

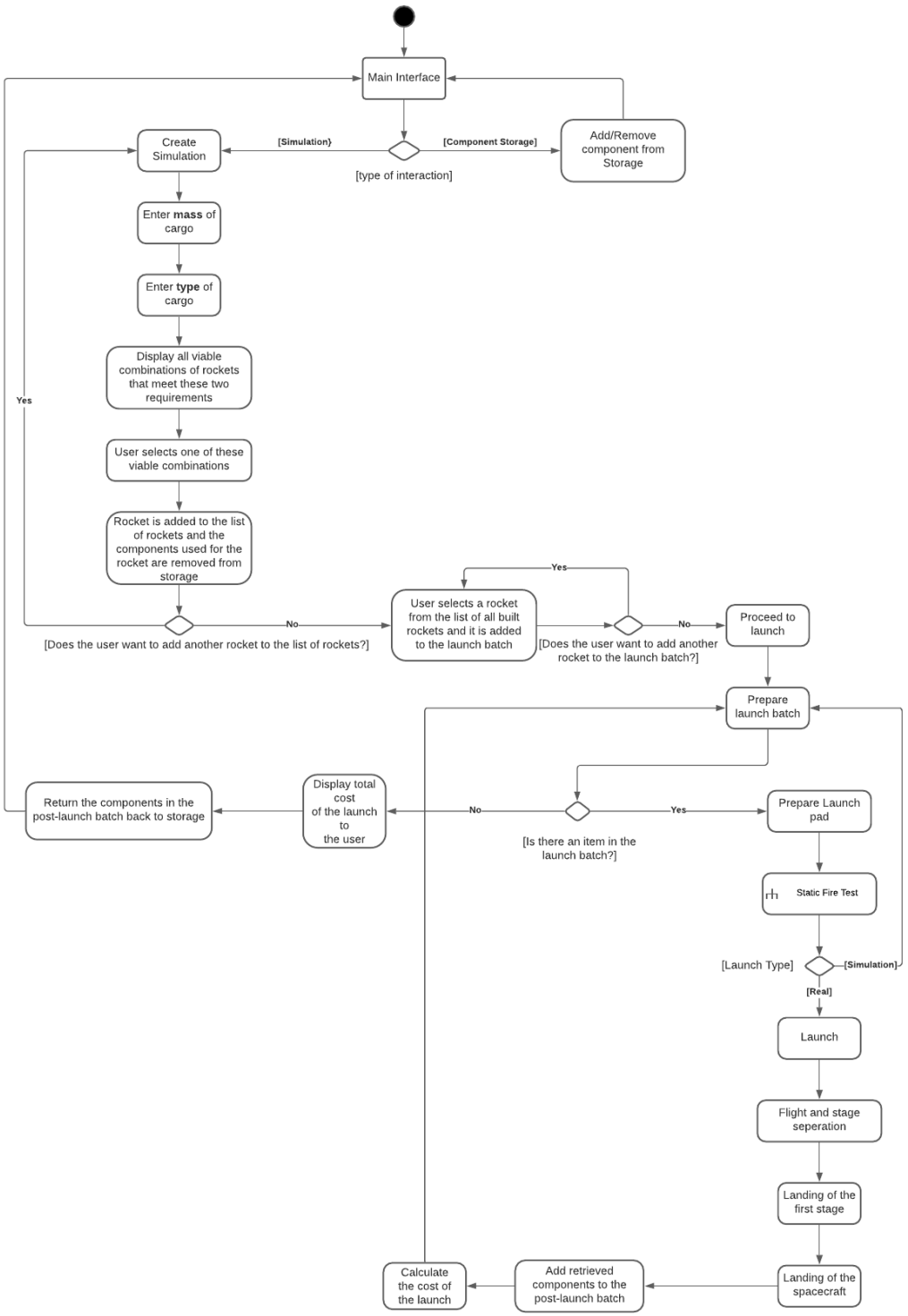
1. The system must be able to ensure that a new launchpad is built or that an existing launchpad has been maintained and is in a condition to withstand the rocket launch.
    1.1. All necessary launchpad inspections should have been performed by a launchpad inspector.
    1.2. If a launchpad that is intended for use has not been inspected, then the user should be notified and given the choice to either send a team in to inspect it or choose another launchpad to use.
    1.3. If the inspection team finds an issue with a launchpad, the user should be notified that it is not usable and another launchpad should be used.
2. The system must be able to run a ”static fire” test by firing up each of the rocket engines to ensure that they are working.
3. The system must provide an interface to set up and run actual launch simulations (where ’actual launch simulations’ refers to launch simulations not being run in test mode).
    3.1. The system should be able to simulate the rocket flight and the separation of stages.
    3.2. The system should be able to simulate a landing of Stage 1 on a drone ship in the middle of the ocean so that it can be refurbished and then reused.
    3.3. The system should be able to simulate:
        3.3.1. A Dragon Spacecraft landing on the International Space Station (ISS).
        3.3.2. The Starlink satellites being launched into LEO and communicating.
        3.3.3. An unsuccessful launch, wherein a component fails.

Diagrams

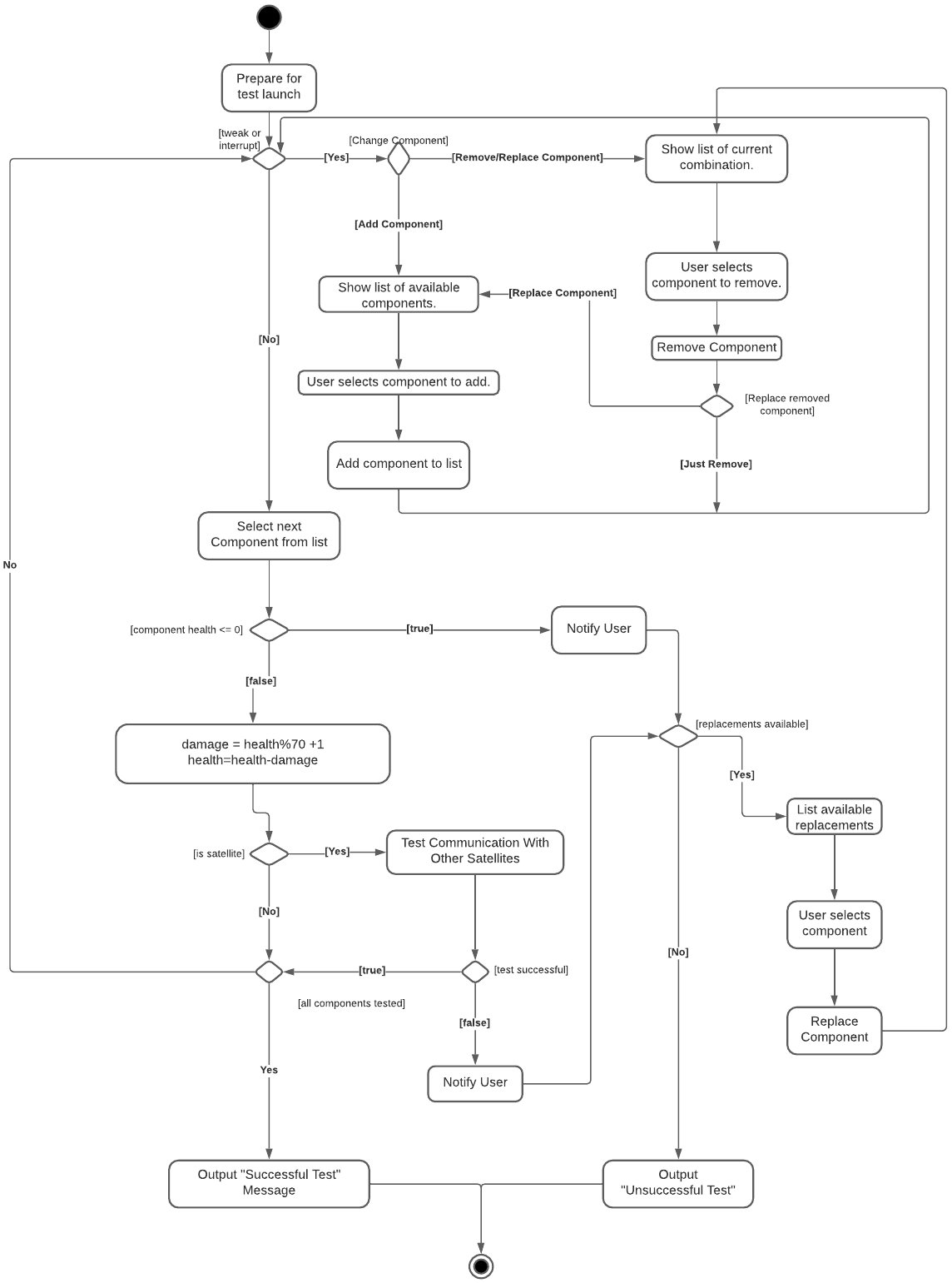
All diagrams were also submitted as a separate PDF so that they can be more easily read.

Activity Diagram

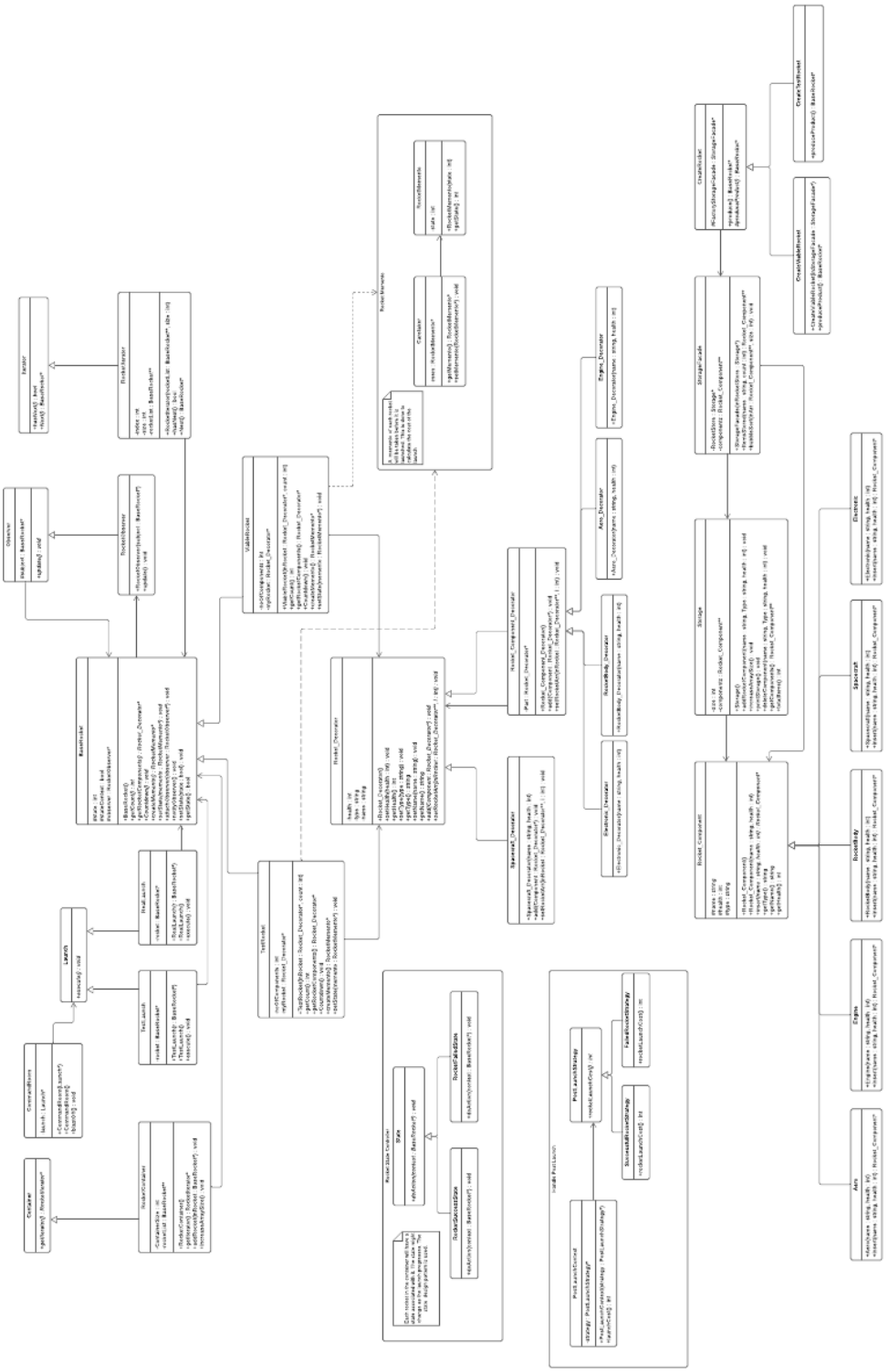
Overall Program



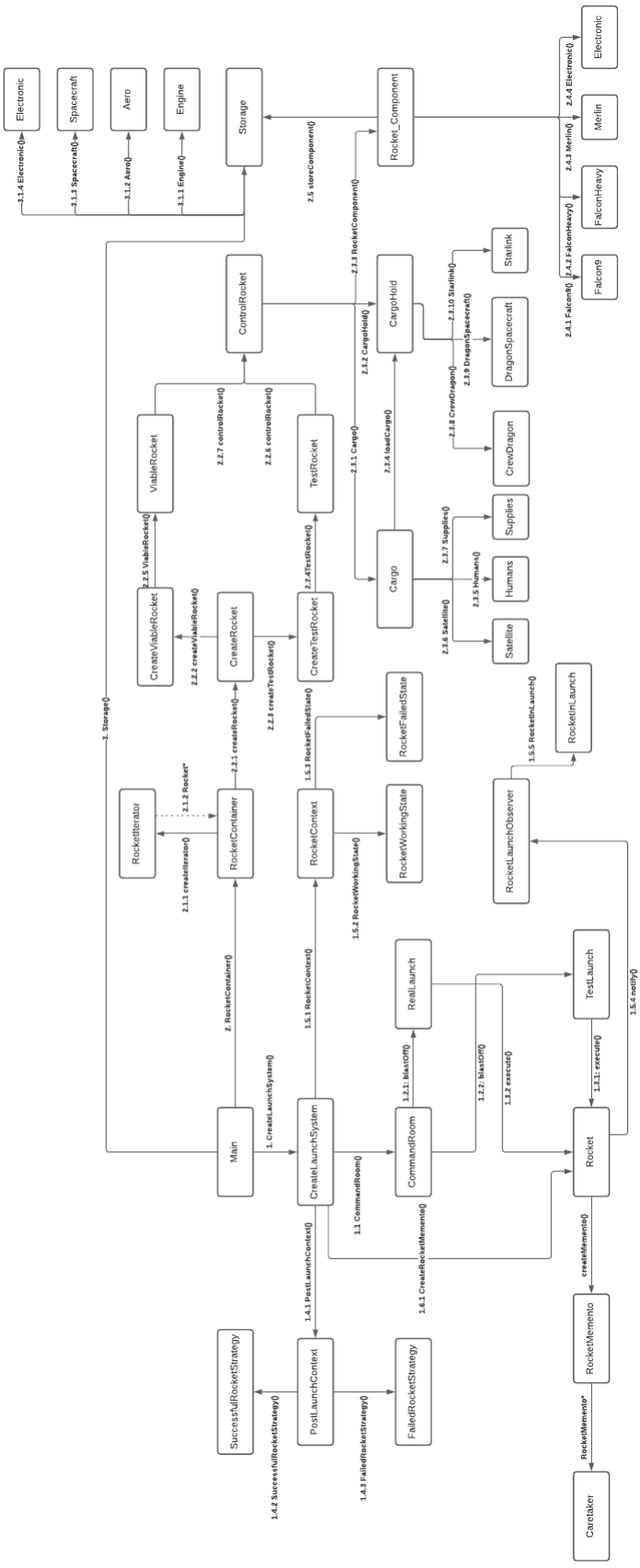
Test Launch


Prepare for test launch
[tweak or interrupt]
[Change Component]
[Yes]
[Remove/Replace Component]
Show list of current combination.
[Add Component]
User selects component to remove.
Show list of available components.
[Replace Component]
[No]
Remove Component
User selects component to add.
[Replace removed component]
Add component to list
[Just Remove]
Select next Component from list
No
[component health <= 0]
[true]
Notify User
[false]
[replacements available]
damage = health%70 +1
health=health-damage
[Yes]
List available replacements
[is satellite]
[Yes]
Test Communication With Other Satellites
User selects component
[No]
[No]
[true]
[test successful]
[all components tested]
[false]
Replace Component
Yes
Notify User
Output "Successful Test" Message
Output "Unsuccessful Test"

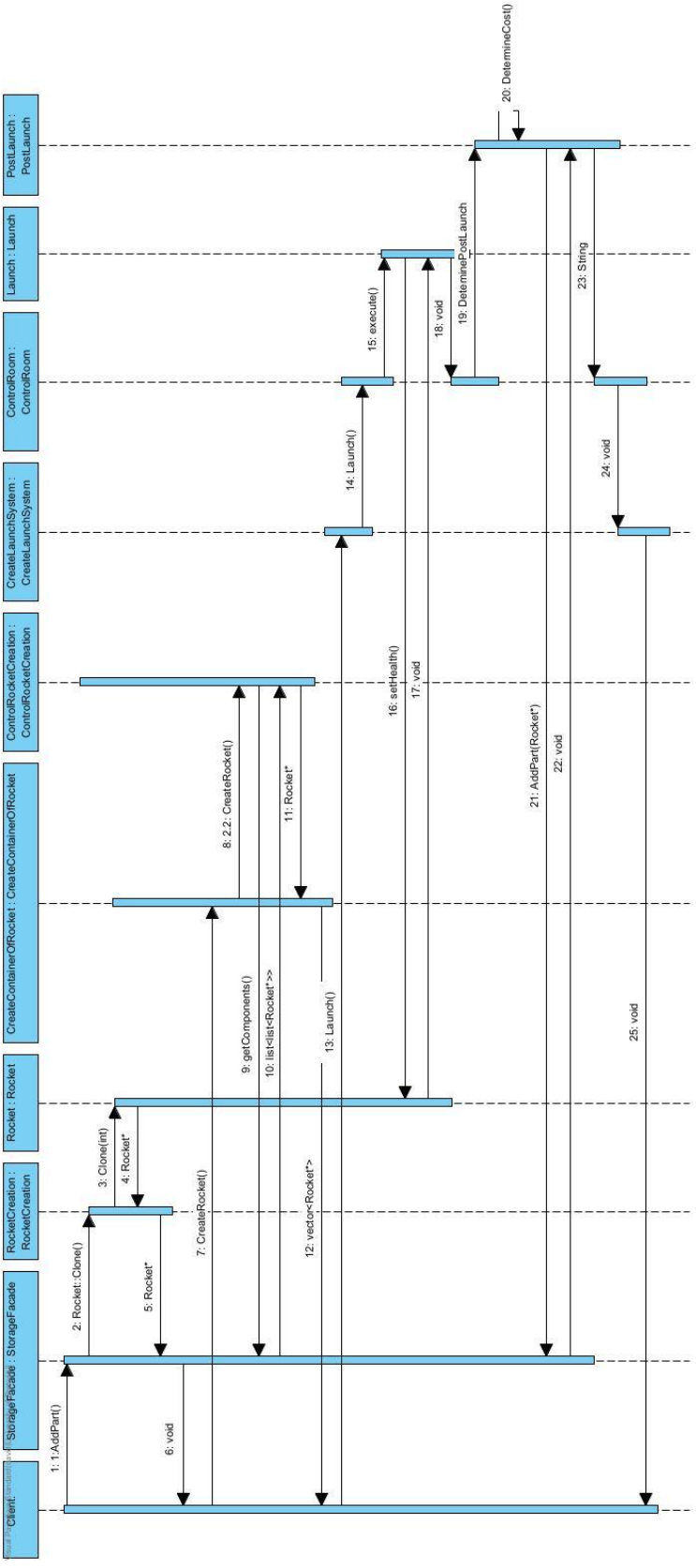
Class Diagram



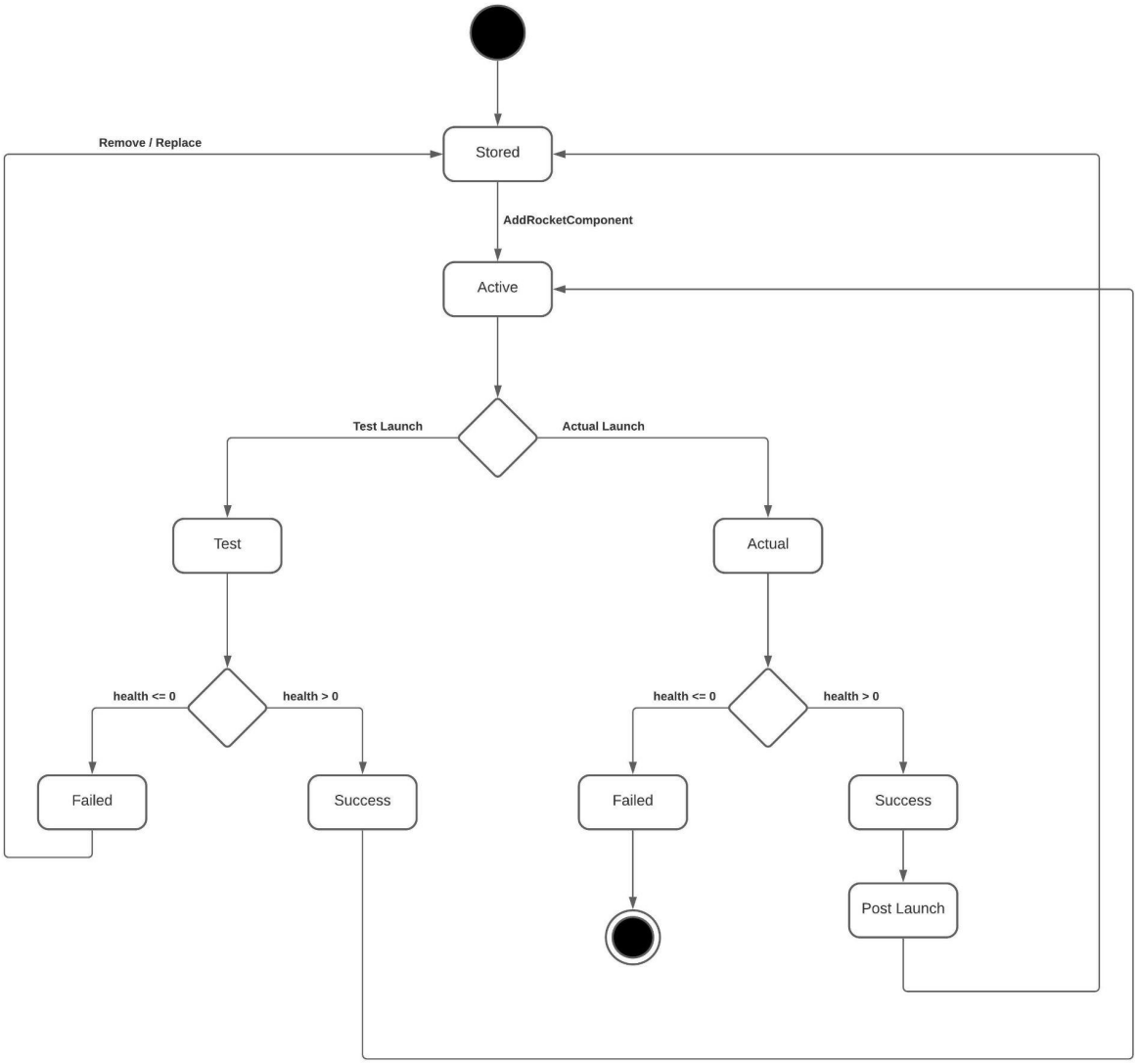
Communication Diagram



Sequence Diagram



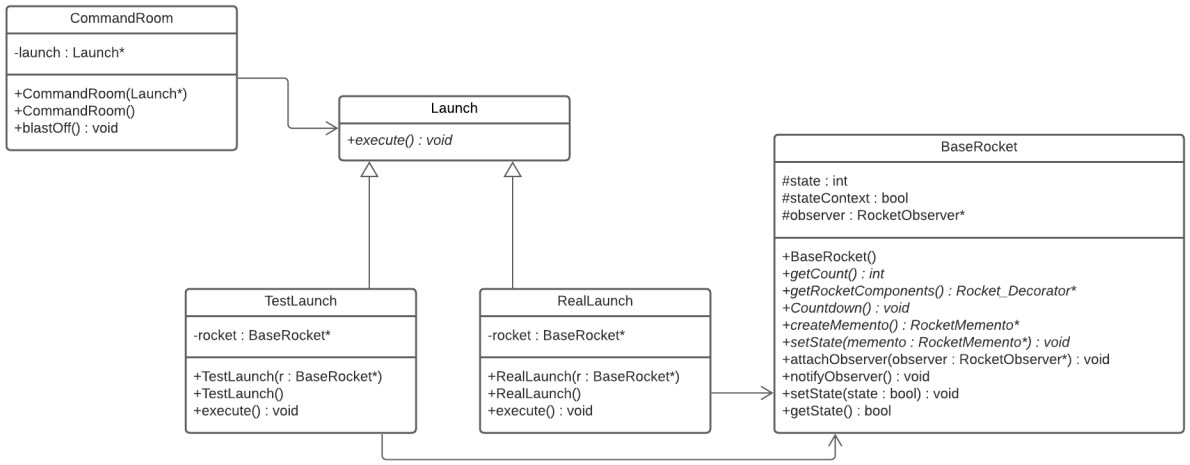
State Diagram (Rocket Component)



Design Patterns - Use and Implementation

Command

UML Class Diagram



Description

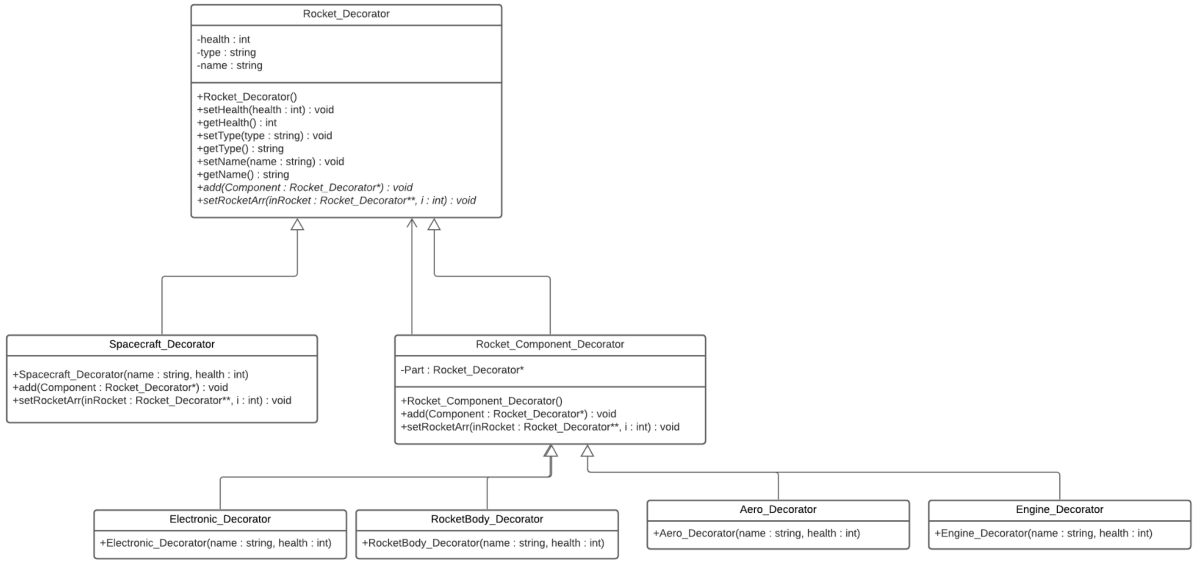
The command design pattern was used to launch rockets from one place using one of the two available commands, this made launching easier and made the main less cluttered. The command is used to execute the main function of the rocket that would send the rocket to space/test the rocket. In the main, a CommandRoom would be instantiated with a command. The CommandRoom would then call its function to execute the relevant function of the rocket related to the specific command. The countdown function would be of type viableRocket or type testRocket depending on what type of rocket was created (real or test rocket).

Participants

Participants	**Class Name**
Invoker	CommandRoom
Receiver	BaseRocket
Command	Launch
ConcreteCommand	RealLaunch
ConcreteCommand	TestLaunch

Decorator

UML Class Diagram



Description

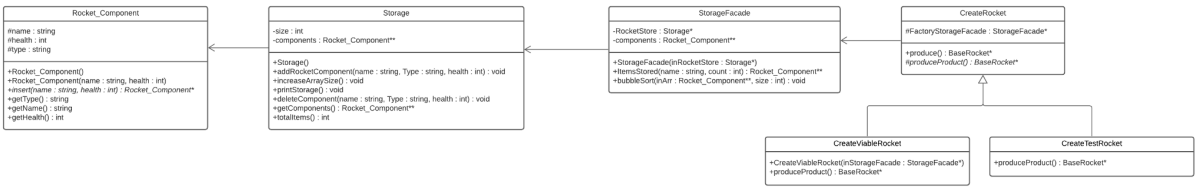
The decorator design pattern was used to build rockets out of multiple different rocket components. This design allowed us to have access to the different components of any rocket at every subsystem in our system. We were able to easily determine which of the components used on the rocket might have failed during launch and to assign “damage” values to each of the components individually. It also allowed us to easily swap out individual components of rockets with ones that were available in our storage.

Participants

Participants	**Class Name**
Component	BaseRocket
Concrete Component	TestRocket, ViableRocket
Decorator	Rocket_Decorator
Concrete Decorator	Spacecraft_Decorator, Rocket_Component_Decorator

Facade

UML Class Diagram



Description

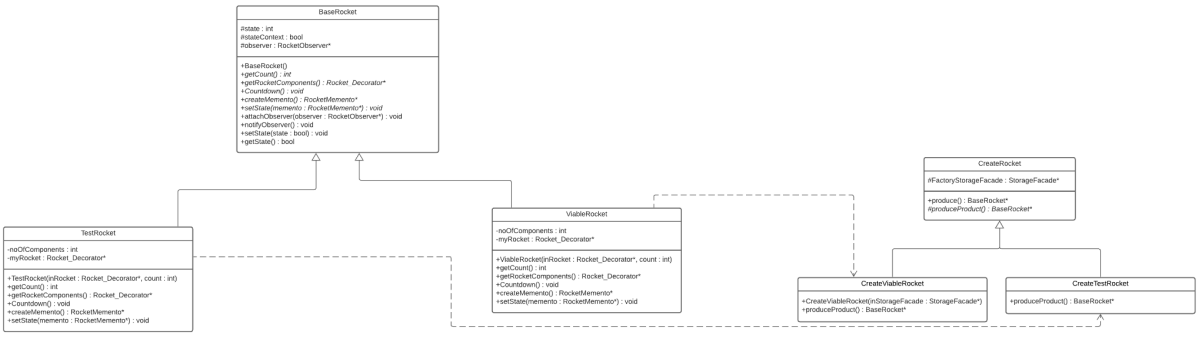
The facade design pattern allowed us to create a generalised interface to our storage subsystem. Since our storage subsystem interacted with the majority of the subsystems in our design, we had to limit the complexity required to interact with it in order to prevent it from having a negative impact on the rest of our system by making our design unnecessarily complicated. It greatly reduced potential clutter, and streamlined all interaction with our storage subsystem.

Participants

Participants	**Class Name**
Facade	StorageFacade
Subsystem classes	Storage, Rocket_Component, Aero, Engine, RocketBody, Spacecraft, Electronic

Factory Method

UML Class Diagram



Description

The factory method was used in combination with the decorator design pattern, and prototype design pattern, to assemble individual rockets out of components that we had in our storage subsystem. It also enabled us to assemble “test rockets” with components that we did not have in our storage subsystem by having the user of our system specify the

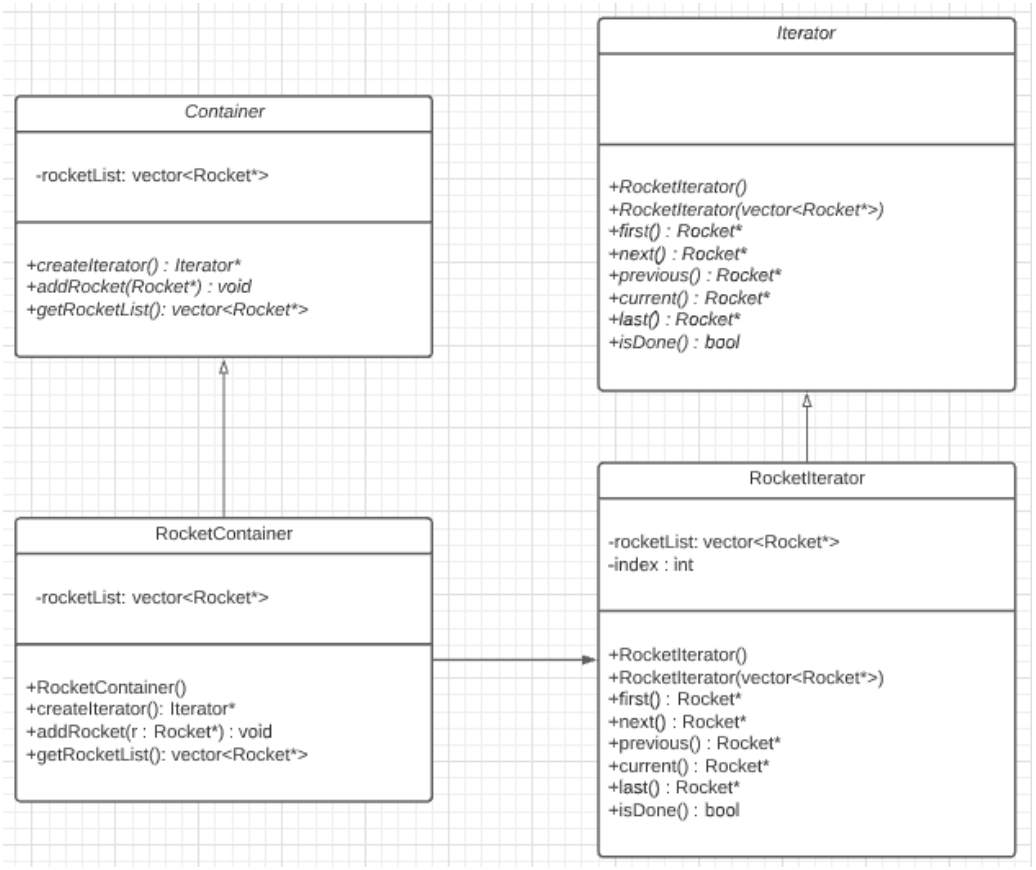
individual components of the rocket. It is central in the creation of all rockets used in our design.

Participants

Participants	**Class Name**
Creator	CreateRocket
Concrete Creator	CreateViable Rocket, CreateTestRocket
Product	BaseRocket
Concrete Product	ViableRocket, TestRocket

Iterator

UML Class Diagram



Description

Our system makes use of a container to group the rockets that are built and ready to launch. These rockets are stored in a vector inside a RocketContainer object. The iterator design

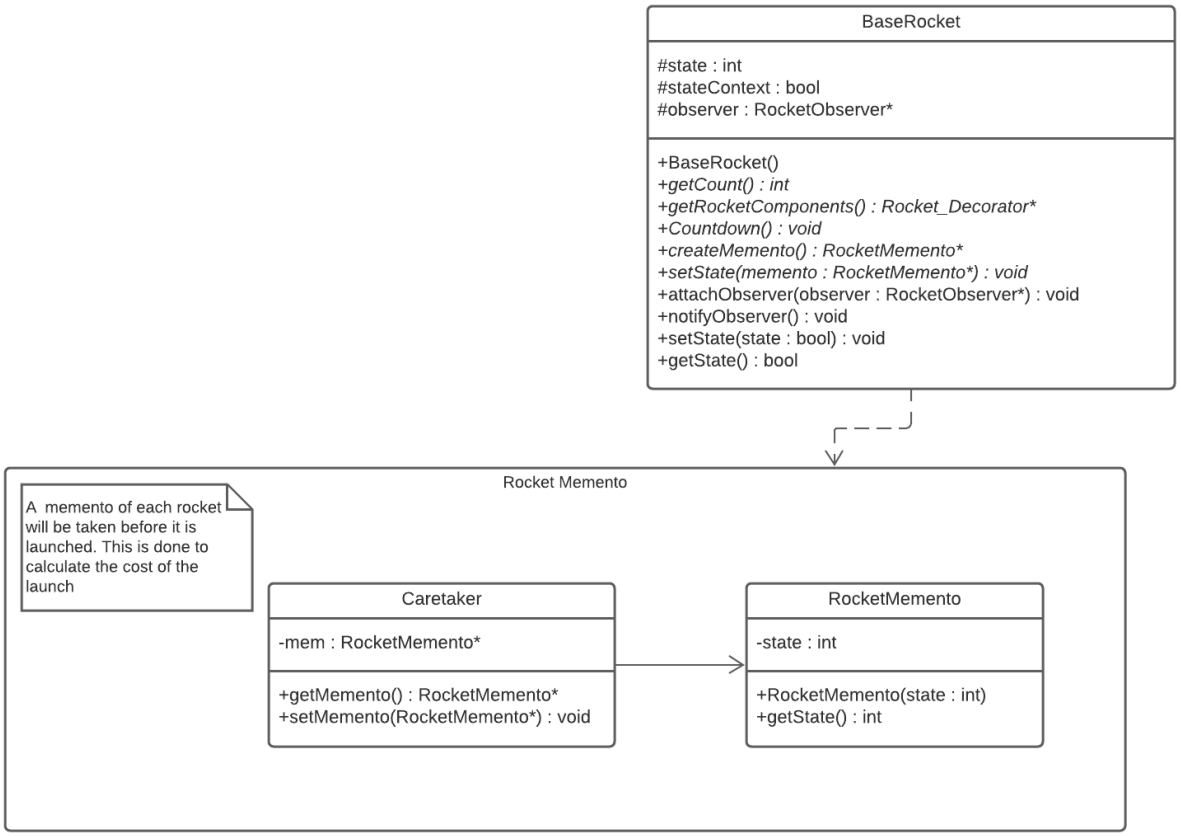
pattern is then used to simplify iteration through the vector to be able to access individual rockets in the batch.

Participants

Participant	**Class Name**
Aggregate	Container
ConcreteAggregate	RocketContainer
Iterator	Iterator
ConcreteIterator	RocketIterator

Memento

UML Class Diagram



Description

The Memento design pattern is used to save the starting state of the current rocket being launched. Once the rocket has been launched and has gone through various conditions, the memento will be restored and compared with the current rocket's state. This comparison will

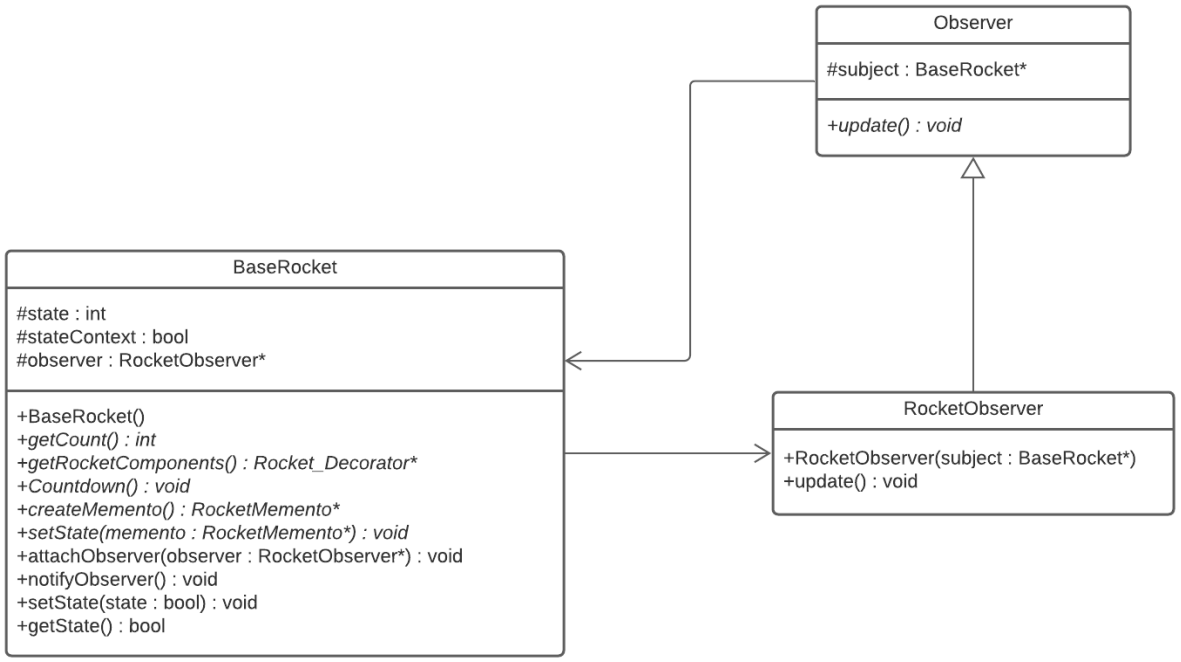
be used to determine if any damage was done which will help determine the costs incurred while launching the rocket.

Participants

Participant	**Class Name**
Originator	Rocket
Memento	RocketMemento
Caretaker	Caretaker
State	State

Observer

UML Class Diagram



Description

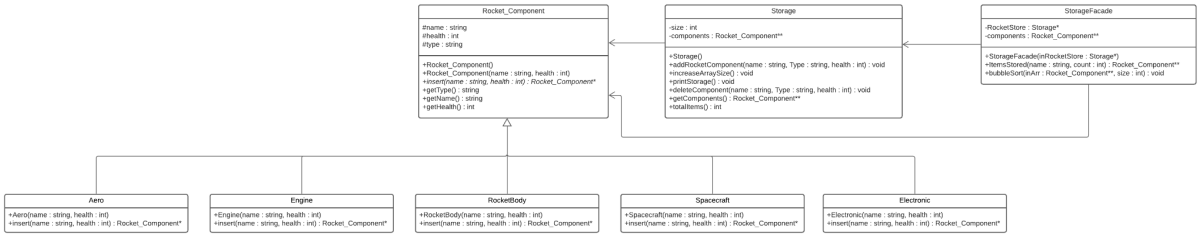
The observer design pattern was used to monitor the progress of rockets that were launched in our system. Every launched rocket in our system has a chance of failing that increases as the health value associated with its components goes down. The observer design pattern enabled us to monitor the progress of each component, and to change the state of the entire rocket in the event that one of its components failed.

Participants

Participants	Class Name
Subject	BaseRocket
Concrete Subject	TestRocket, ViableRocket
Observer	Observer
Concrete Observer	RocketObserver

Prototype

UML Class Diagram



Description

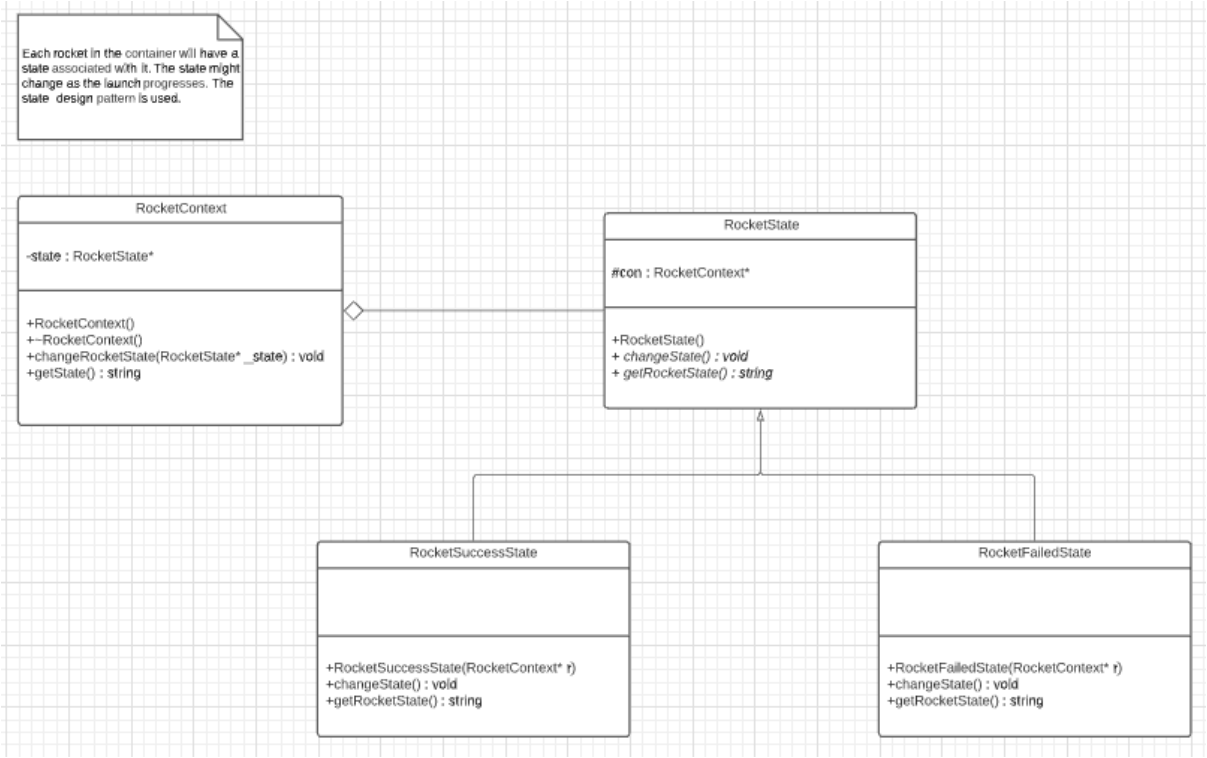
The prototype design pattern was used in our storage subsystem. It allowed us to add rocket components to storage by making clones of the components that we already had. Since both the falcon 9 and falcon heavy rockets make use of the same engines, electronic components and aerodynamic components, only 5 different types of components have to be stored that we can use to clone components for all of the rockets assembled by our system.

Participants

Participants	Class Name
Prototype	Rocket_Component
Concrete Prototype	Aero, Engine, RocketBody, Spacecraft, Electronic
Client	Storage

State

UML Class Diagram



Description

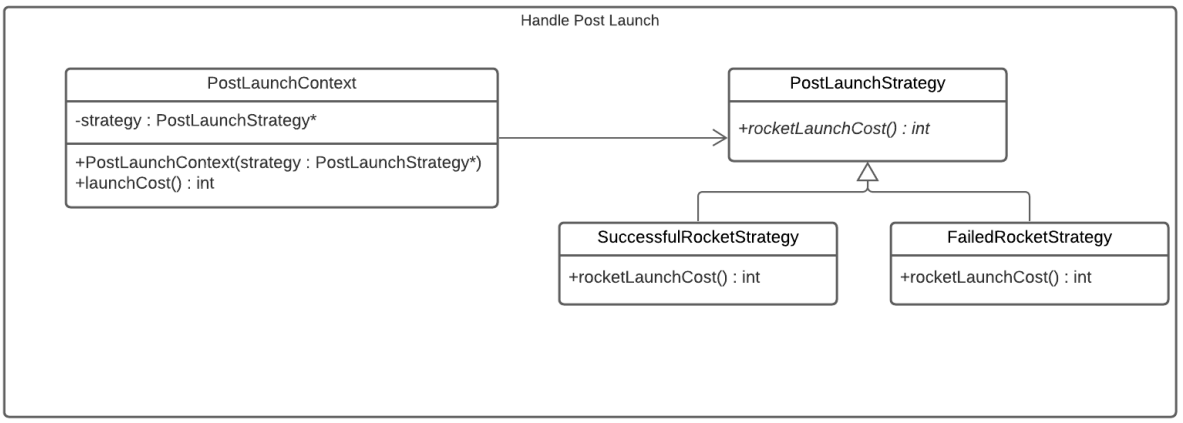
In order to properly define launch simulation outcomes, we use the state pattern to effectively model the different states during/after a launch. The different states a rocket can be in after a launch are a successful state and a failed state.

Participants

Participant	**Class Name**
Context	RocketContext
State	RocketState
ConcreteState’s	RocketSuccessState, RocketFailedState

Strategy

UML Class Diagram



Description

The Strategy Design Pattern is used to execute the operations that must take place once the launch is complete- returning the launch cost and restoring the rocket components to storage on a successful launch.

Participants

Participant	**Class Name**
Context	PostLaunchContext
Strategy	PostLaunchStrategy
Concrete Strategy	SuccessfulRocketStrategy, FailedRocketStrategy