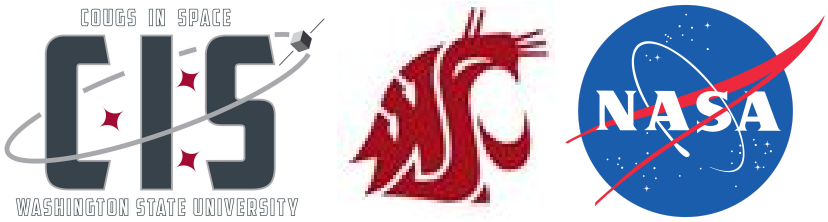
# CougSat Simulation Software

## Project Requirements Specifications





### Mentor

Aaron Crandall

### Team Viking

Courtney Snyder
Angie Park
Solomon Egwuonwu


Course: CptS 421 Software Design Project I
Instructor: Aaron Crandall

# TABLE OF CONTENTS

# I. Introduction

Our client, the club Cougs in Space of Washington State University (WSU) in Pullman, wants to know that their cubesat, CougSat I, will successfully launch and orbit. To help them feel more confident and help them further develop their design, we will build a testing suite for them. This testing suite will go through all stages of both the launch and orbit to determine if anything could possibly go wrong.
Development of the testing suite will be broken up into two phases, the alpha prototype and the beta prototype.
The alpha prototype will be a state machine that looks at all of the top-level behaviors that CougSat I will have, most of which will be represented by various power modes.
The beta prototype will be a functional testing suite that behaves as the satellite itself and receives real-time inputs from physical testing, or allows users to manually enter the inputs.
Section II discusses the system requirements, which includes use cases as well as functional and nonfunctional requirements.
Section III discusses the system evolution, which includes the fundamental assumptions incorporated in the system design, as well as how anticipated changes and additional features will be added to the system in the future.

# II. System Requirements Specification

We have been working on contacting the client and asking various team leads for requirements and specifications, and have encountered multiple issues that all trace back to the fact that they do not know what they require of our software yet, as they have just started building. We will get some expected inputs and outputs from each sub-team lead within the next week; with these we will be able to better model our state machine after CougSat I. Most importantly, we have seen just how crucial our software will be in pushing the construction process forward, as we are asking people to answer questions that they were not thinking of, and giving them a tool to come up with initial requirements as well as add additional requirements later on.

## II.1. Use Cases

These use cases are based on user behavior and the system behavior.

### II.1.1. Successful launch use case

**Actors:** CougSat I, Cougs in Space ground station, Cougs in Space

**Description:** CougSat I after released in orbit remain asleep for 30 minutes after which it’s ready for operation.

**Stimulus:** CougSat I establish communication link with the ground station and requests transmission of data.

**Response:** The input data, real-time or pre-written is sent to CougSat I.

**Comments:** The simulator will determine if, given inputs, CougSat I will be successfully launched and orbit.

### II.1.2. Report status use case

**Actors:** CougSat I, Cougs in Space ground station, Cougs in Space

**Description:** CougSat I sends a summary of its status to the ground station. This includes details of its instruments as tested and any other information about potential problems that have been detected.

**Stimulus:** The ground station establishes a communication link with the CougSat I and request transmission of status data.

**Response:** The CougSat I status data is sent to the ground station.

**Comments:** The ground station, Cougs in Space and WSU are constantly monitoring for any abnormality, power shortage and output data.

### II.1.3. Low power mode use case

**Actors:** CougSat I, Cougs in Space ground station, Cougs in Space.

**Description:** The effect of this command is to run CougSat I in a minimal energy state as to conserve energy. Data collection is turned off; the only subsystems that are on are the main board, the power board, the attitude control board, the communication board, and the location module.

**Stimulus:** CougSat I establishes a communication link with the ground station and request to move to power saving mode.

**Response:** The ground station should acknowledge that CougSat I is in a power saving state.

**Comment:** This should be expected when the CougSat I goes into an area without sunlight for a long period of time, or when the battery is between 25% and 50%.

### II.1.4. Reboot use case

**Actors:** CougSat I, Cougs in Space ground station, Cougs in Space.

**Description:** CougSat I should be able to go into a reboot state. In the reboot state, all instruments to turn off and turn back on.

**Stimulus:** Initial CougSat I power up and any other necessary restarting of various subsystems; for instance, rebooting all data collection modules when transitioning from Emergency mode to Low Power mode.

**Response:** CougSat I should reboot and notify the ground station what has been rebooted and why.

**Comments:** All subsystem reboot should be rare. Otherwise, it is only required after the system reboots individual subsystems..

### II.1.5. Requirements use case

**Actors:** CougSat I, Cougs in Space ground station, Cougs in Space.

**Description:** CougSat I should meet and adhere to NASA's regulations and specifications in order to launch.

**Stimulus:** All aspect of from building to launch must meet all given NASA requirements. Our system will be built primarily around these requirements, with input from the client if it does not contradict the given requirements.

**Response:** The simulator should determine whether the given hardware and behaviors of CougSat I meet NASA's requirements and specifications.

**Comment:** This is a very important aspect of building the CougSat I and should be taken seriously, as without launching CougSat I, there is no orbit, and the mission ends before it begins.

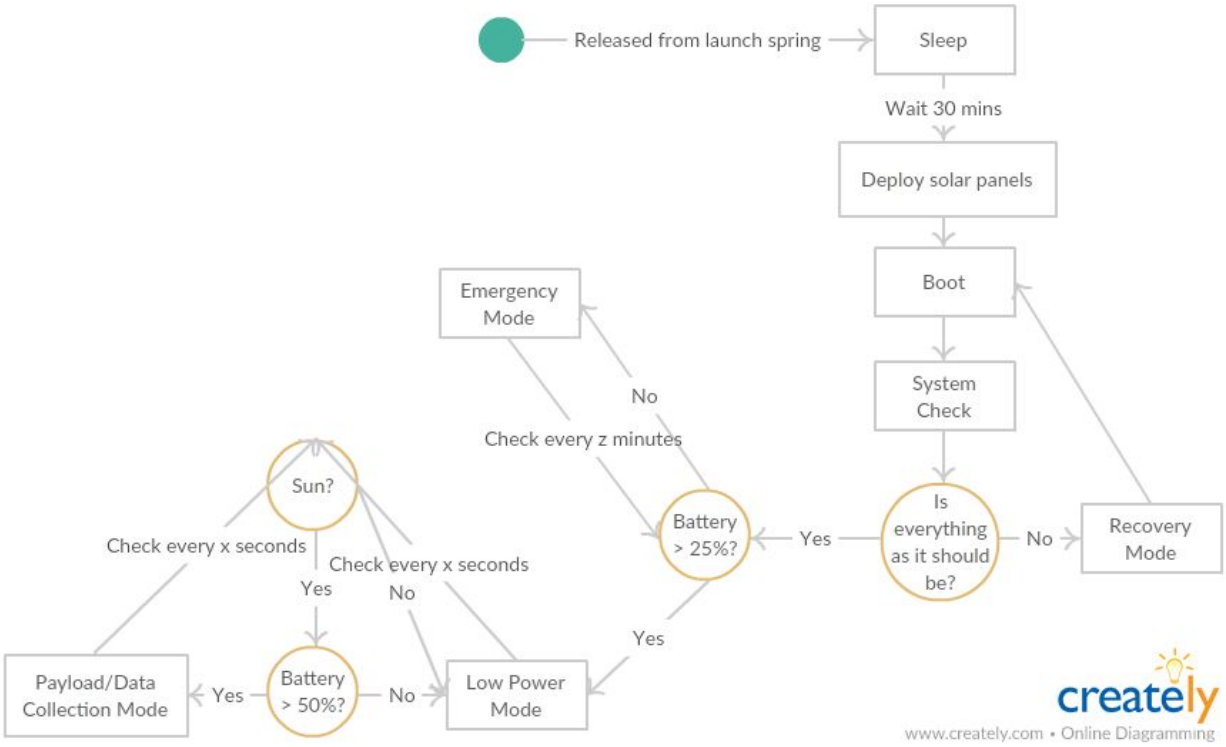



*Figure 1. CougSat I Behavioral State Machine, version 2.0*

# II.2. Functional Requirements

## II.2.1. Support multiple input types

**Ability to model real-time and pre-written inputs:** Our simulator will be able to take both real-time data or an input file with valid sensor inputs and produce a realistic simulation of CougSat I. Real-time data is from the satellite sensors collecting data while it is being physically tested, and the input file will mock that sensor data collection by providing raw numbers.
**Source**: This requirement was inspired by another on-campus club called Palouse RoboSub club. Their simulator behaves in a very similar manner and is very accurate and user friendly. Our client needs an accurate simulator to help them determine what parts to use, whether those parts can be integrated, and how likely those parts are to fail or cause other problems in orbit.

## II.2.2 Unit testing

**Accurate sub-system modeling for unit testing:** Each subsystem board must behave in such a way that the board does not monopolize the power source and communicates appropriately with the main board. We must accurately model what these behaviors should look like so that if something is wrong with one of the boards, it can be rectified.
**Source**: Our client needs a way to represent their subsystem circuit boards, as well as the software on those boards. Our simulator will use the outputs of the hardware and the software

to model the behavior of each subsystem. Also, our client needs to integration test their subsystems, and our software can only do that if each subsystem is accurately simulated.

### II.2.3. Integration testing

**Determine if sub-system integration is successful:** Using our simulator and giving it inputs, the client will be able to tell what subsystem caused CougSat I to fail and what input caused it to fail. The client can then used that data to adjust those components accordingly.
**Source**: This requirement is one of the main premises of our project, and is the main premise of our software. Our client needs to be confident that CougSat I will behave appropriately during launch and orbit to ensure a successful mission.

### II.2.4. System testing

**Confirm that the testing suite works and satellite behaviors follow NASA regulations:** The whole testing suite needs to work; the unit testing and integration testing, in order to simulate the launch and orbit. But, our client needs the CougSat I to launch in order for it to do much of anything, and in order to confirm that it will be able to launch, we need to include NASA's cubesat requirements into our system.
**Source**: This requirement is crucial to successfully launching CougSat I, so it is necessary to include this in our testing suite.

### II.2.5. Acceptance testing

**Ability to tell if the satellite will successfully launch and orbit:** Using our simulator and giving it inputs, the client will be able to tell what subsystem caused CougSat I to fail and what input caused it to fail. The client can then used that data to adjust those components accordingly.
**Source**: This requirement is one of the main premises of our project, and is the main premise of our software. Our client needs to be confident that CougSat I will behave appropriately during launch and orbit to ensure a successful mission.

## II.3. Non-Functional Requirements

**Execution accuracy**: The simulator as a whole needs to be as accurate as possible, so when it is given inputs, it can simulate the launch and orbit of CougSat I as accurately as possible and alert the client if anything will fail or cause problems in space so that they can fix it.
**Reliability**: The simulator needs to be as reliable as possible so that the client can use it whenever they need and when they use it, and it does not freeze or crash on them.
**Input capability**: The simulator needs to accept both real-time and pre-written data as inputs.

**Output capability**: The simulator must return the state of the launch, success or failure, and a log of what errors or failures occurred during the simulation, what inputs caused those errors.
**Bad data trapping**: Because the simulator accepts pre-written data as inputs, it must be able to sense when data is invalid; that is, out of bounds (eg temperature reading is too high for the sensor to capture) or physically impossible (eg sun sensor reading is a large positive number when the satellite is on the dark side of the earth). Once sensing the bad data, the simulator must alert the user that it cannot simulate a launch with invalid data, and provide what the invalid data is.
**Accurate data recording**: When a board failure occurs, the simulator needs to be able to be accurate and detailed when recording the failure in the failure log so that the client can know exactly what caused the failure.

# III. System Evolution

We must make some assumptions before implementing our testing suite; we have confirmed with the club that the following assumptions are appropriate to make.
First, the satellite contains a battery level indicator. This is important because multiple transitions between behavior modes on the satellite depend on the power level; emergency mode, low power mode, and data collection mode. Emergency mode is for when the satellite gets into self-charging state when its battery goes below 25%, low power mode keeps the necessary controls on communications, power, attitude, and location are on for when the battery level is between 25% and 50%, and data collection mode for taking pictures of various objects when the battery level is above 50%. In addition, for our testing simulator, it would be challenging if we cannot get a numerical value of battery, as we would not easily be able to tell if the battery on the satellite died.
Second assumption we made is that the satellite has at least one light sensor. As Cougs in Space already determined, CougSat I will have solar panels on it. The light sensor would perform some important tasks on the solar panels. For testing the satellite's behavior, we need to put the diverse condition it can encounter in the space, and the light sensor would let us configure some testing cases with/without the sunlight.
It is unclear if the club would like deployable solar panels or solar panels place directly on the satellite, though we have made an assumption on their behavior. We have assumed that if the solar panels are deployable, they only deploy once before the satellite system is booted up and they do not fold back in.
The fourth assumption our project is based on is that CougSat I's top-level payload is a camera. With this assumption, we also can assume the data it collects and both when and how the camera should work. The data will be mostly pictures, and its work will be focused on taking pictures, even though the decision of what kind of pictures it will take has not been made yet.
With these assumptions, we can implement the testing across the features of the satellite. For example, we have to check the camera working time matches the time that the light sensor says there is a sunlight on the satellite.

Another assumption is about communication. After the satellite is launched, it will be tumbling in space; to counteract this, the satellite will need to go into a detumbling state. Whether the satellite goes into this state before or after the transmission of information is unknown, but will become more clear as antenna design progresses. The assumption of having an initial detumbling state ensures that the antenna of CougSat can point where it needs to so that transmissions between the satellite and the ground station will be more efficient and have a lower chance of getting any potential jamming. To be more specific, the satellite will only enter the detumbling state once before it gets into the proper orbit. After getting into an initial orbit, the attitude board will only be used for adjusting the satellite's position as necessary. On our project, testing for detumbling when the ground control gets transmission from the satellite will work for checking its capacity.

Lastly, we will let the simulation progress through states until the battery dies, system is fried from heat or radiation, or the system stops functioning for any reasons even after one test fails. As the project is huge, it will take a lot of time to develop each test, identify all potential causes of failure for each test, and repeat the process again. This assumption is most important to acknowledge in the unit testing because if it is not fixed at that level, it will cause problems in the integration testing.

For now, the systemic and functional specifications of CougSat I are not completely determined, so our project will go through a lot of changes. As we see it, NASA's requirements will most likely not change, and to get launched, the club should develop CougSat I primarily around NASA's requirements.

We need to consider future changes to CougSat I. As the club starts to set up their own requirements, we look forward some massive changes in our project as well. This is because their newly developed requirements may impact on the fundamental assumption we already considered above. Our project is building a testing suite for Coug Sat I, so the lack of requirement on the club project greatly influences the direction our project, in regards to adding new functionality or tests, or adjusting existing ones. On the other hand, since our two projects are connected, the application of changes on user needs will most likely not changed.

At the same time, we need to be aware of the second and the third law of Lehman. It says the more complex the system, the more challenging it is to conserve the software's familiarity with the implementation. When the time comes up and we have more and more requirements to apply in our project, the prior considerations should be conserved despite of the increasing complexity, as if nothing is removed, the measurements they test may still be relevant. The continuous input of our client will continue to expand our program, which is connected to the Lehman's 4$^{th}$ law. We expect our project will continuously have new features to test, and the growth of it is inevitable. As a testing simulator, it cannot settle down at certain point and it allows to follow the 5$^{th}$ law of Lehman, "declining quality". Finally, as our project is completely based on CougSat I's behavior during launch and orbit, the project will respond to all inputs it receives and determine if those inputs are dangerous to the mission.

In the evolutionary stage of our project, we will make sure it fulfills the laws of software evolution built by Lehman by asking for detailed requests from our client, and ensuring that any new requests do not contradict prior requests unless the feature has been changed.

# IV. Glossary

**Cubesat**: A cube-shaped satellite commonly used by researchers to collect data. They come in various sizes; 1 U, 2 U, 3 U, and 6 U. Each U is 10 cubic centimeters.
**Alpha prototype**: Used to assess whether product works as intended. Similar in material and geometry as production version, but made differently (Fisher).
**Beta prototype**: Used to assess reliability and to identify remaining bugs in the product. Given to customers for testing in the use environment. Parts are usually made with the actual production process and supplied by intended parts suppliers. Assembled by team rather than the manufacturer (Fisher).
**State machine**: A set of states the satellite can be in and what actions cause the satellite to transition to other states.
**Testing suite**: A single program that can do unit testing, integration testing, system testing, and acceptance testing.

# V. References

Cite your references here.
For the papers you cite give the authors, the title of the article, the journal name, journal volume number, date of publication and inclusive page numbers. Giving only the URL for the journal is not appropriate.
For the websites, give the title, author (if applicable) and the website URL.

Fisher, Andrew. *Prototyping*. Memorial University of Newfoundland. http://www.engr.mun.ca/~adfisher/7936-06/Lectures/Prototyping-HO.pdf

Griffiths, Mike. *Non-Functional Requirements - Minimal Checklist*. Leading Answers. http://leadinganswers.typepad.com/leading_answers/2009/03/nonfunctional-requirements-minimal-checklist.html

Karch, Erich. “Lehman’s Laws of Software Evolution and the Staged-Model.” *Karchworld Identity.* https://blogs.msdn.microsoft.com/karchworld_identity/2011/04/01/lehmans-laws-of-software-evolution-and-the-staged-model/.