CougSat Simulation Software

Project Requirements Specifications

Mentor

Aaron Crandall

Team Viking

Courtney Snyder
Angie Park
Edward Kuo
Solomon Egwuonwu

Course: CptS 423 Software Design Project II
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 is a state machine that looks at all of the top-level behaviors that CougSat I will exhibit.
The beta prototype will be a functional testing suite that simulates CougSat I through launch and orbit, and receives real-time inputs from the user through the simulated ground station.
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. The cases are also dependent on the simulating conditions.

II.1.1. Modifying COLLADA files

Actors: Cougs in Space

Description: The user uses Blender to modify the 3D model of the satellite and he/she uses any kind of text editor to add/remove the properties of the satellite, like solar panel, sensor, etc.

Stimulus: The user updates the config file with the modified COLLADA file and runs the simulator using the updated config file.

Response: When the simulator runs, right-click on CougSat I → Properties → “Model” in 3D Graphics category → Model. The Model file section has the COLLADA file that the user modified.

Comments: This should be documented well, so that the common users do not feel frustrated when they have to deal with unfamiliar format.

II.1.2. Check requirements

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. The state machines we have as our alpha prototype are built on the them. So, CougSat I Simulation Software will integrate the satellite with its behaviors using Python script and the state machines.

Response: The simulator shows the behaviors of CougSat I. It will not be tough to determine whether they meet NASA’s requirements and specifications by seeing the result of the simulator, because once the simulator runs, it means requirements are satisfied.

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.

II.1.3. Successful auto-run simulator

Actors: Cougs in Space, CougSat I, Cougs in Space ground stations

Description: CougSat Simulation Software contains five ground stations and the model of satellite that is orbiting around the Earth.

Stimulus: The user run STK plugin script on python with the their own configuration file containing the orbit, time period, path of COLLADA file for satellite model, and so on.

Response: CougSat I and the ground stations are established in the simulator and the satellite automatically begins running in the orbit and time period that the user set.

Comments: It will be obvious if the simulator is launched successfully and it is running.

II.1.4. Interacting with satellite

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

Description: CougSat I should be able to interact with the ground stations of Cougs in Space. In the case of interaction, Cougs in Space asks CougSat I's status and it should respond back.

Stimulus: It simulates the use case that the user sends signal to the satellite through the ground stations. The user should input the certain command to see the result of the interaction.

Response: There is an output file (.csv). The users will be able to see if the satellite understands their commands on the specific time.

Comments: The output file should be organized, as it contains many results other than the interaction as well.

II.2. Functional Requirements

II.2.1. Support given input types

Ability to model a given input: Our simulator will be able to take an input file with valid sensor inputs and produce a realistic simulation of CougSat I. The input file will mock the 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.
Priority: priority level 1: Essential and required priority.

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.
Priority: priority level 1: Essential and required priority.

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.
Priority: priority level 1: Essential and required priority.

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.
Priority: priority level 1: Essential and required priority.

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.
Priority: priority level 1: Essential and required priority.

II.3. Non-Functional Requirements

Execution accuracy: The simulator 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. The

simulator will alert the client if anything fails or causes problems while running so that it can be fixed.
Storage: The computer(s) that will be using the simulator must be able to have enough memory to store STK and the input files being executed.
Scalability: The simulator will be able to run on multiple computers as long as there is an acceptable input for each instance.
Reliability: The simulator needs to be as reliable as possible so that the client can use it whenever they need to. This means that when the client is using it, the simulator does not freeze or crash on them.
Input capability: The simulator needs to accept both real-time and pre-written data as inputs. The input data will be in the form of a pre-written config file or prompted by the user by the program. Python must be downloaded on the computer to execute. If any input data is bad, then the simulator must provide feedback and exit.
Output capability: The simulator must return the state of the launch, success or failure. Others include a log of what errors or failures occurred during the simulation, what inputs caused those errors if applicable, and the simulator must be able to return data when prompted to by the ground station/user during execution.
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. The simulator will then provide feedback to what the invalid data is and the acceptable range of data that can be inputted.
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. This will allow the client to 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. We must also think about the ramifications of our design choices in the scope of other engineers using this software as a reference for their design. The software needs to be precise enough to precisely simulate CougSat I in space, but flexible enough to allow users to update physical and behavioral traits as needed without losing any of the accuracy.
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 and the simulation was over.

The second assumption we made is that the satellite has at least one light sensor. As Cougs in Space already determined, CougSat I will have multiple solar panels on it. The light sensor is necessary because it will indicate whether the satellite is in the light or not, and therefore whether it has the ability to collect light. For testing the satellite's behavior, we need to account for the diverse conditions it can encounter in the space, and the light sensor will let us configure some testing cases with and without the sunlight. It is still unclear if the club would like deployable solar panels or solar panels place directly on the satellite, though we have made an assumption about each of these behaviors. 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 into the satellite.

The third 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. Though we have included the payload circuit board in the state machine, we do not have a circuit board design for this component, so we have not expanded on the functionality.

Our next assumption is that the documentation we provide will be enough for Cougs in Space members to adjust and expand upon our design. For instance, adding the camera to the COLLADA file and adding the camera functionality to the empty payload board class. This will also help Cougs in Space add more features to the physical satellite or its state machine behaviors, which could be useful for modelling CougSat II.

Another assumption is about communication with the ground station. After the satellite is launched, it will be tumbling in space; to counteract this, the satellite will need to go into a detumbling state. The satellite will go into this state before the transmission of information, as the antenna will extend out of the cube and may snap off if it does this while the satellite is tumbling. The assumption of having an initial detumbling state also 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 or the simulation time period specified by the user is over.

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 simulator 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 CougSat 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 change.
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 4th 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 5th 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

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).
COLLADA: COLLADA is an XML-based schema that makes it easy to transport 3D assets between applications, enabling diverse 3D authoring and content processing tools. The intermediate language provides comprehensive encoding of visual scenes including: geometry, shaders and effects, physics, animation, kinematics, and even multiple version representations of the same asset (Khronos).
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.
LEO: Low Earth Orbit is an orbit around Earth with an altitude of 2,000 km (1,200 mi) or less, and with an orbital period of between about 84 and 127 minutes.
Simulation: For our purposes, a simulation is the combined behavior of the Earth, the cubesat, and the ground station.

State machine: A set of states the satellite can be in and what actions cause the satellite to transition to other states.
STK: Systems Tool Kit is an environment that accurately represents physics on Earth and in LEO.
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/.

Khronos. *COLLADA Overview.* https://www.khronos.org/collada/