Software Requirements Specification

for

Satellite Anomaly Injection & Detection Testbed

Version 1.0 Approved

Jerome Pineda

Matthew Gilligan

[illegible]

Prepared by Alex Huang, Jerome Pineda, Samantha Simpson

CSULA / The Aerospace Corporation

December 7, 2020

Table of Contents

Revision History

Name	Date	Reason for Changes	Version
Requirements Committee	9/8/2020	Initial Document: delivery of software requirements	Draft #1
Requirements Committee	9/28/2020	Revisions Post Introduction Meeting	Draft #2
Requirements Committee	10/5/2020	Revisions to be made post feedback from sharing with the entire team	Draft #3
Requirements Committee	10/21/2020	Revisions made post general team meeting on 10/16	Draft #4
Requirements Committee	10/23/2020	Revisions made post feedback on 10/23	Draft #5
Requirements Committee	10/27/2020	Revisions made post feedback on 10/27	Draft #6
Requirements Committee	11/14/2020	Revisions made post feedback on 11/14	Draft #7
Requirements Committee	12/3/2020	Revisions made post feedback on 12/3	Version 1.0

1. Introduction

1.1 Purpose

The purpose of this document is to supply information on the anomaly injection and detection requirements. Also, providing information on simulation tools allows the users to configure and deploy platforms used in a real-time environment to simulate anomalies.

1.2 Intended Audience and Reading Suggestions

This Software Requirements document is intentional for:

- Software Developers can review the project's capabilities and more easily understand where their efforts should target to improve or add more features to it.
- Project testers can use this document as a base for their testing strategy as some bugs are easier to find using a requirements document. This way of testing becomes more methodically organized.
- Project reviewers can use this document as a base for analyzing and critiquing design plans.
- Project managers can use this document to help the project team plan how they should implement its quality control.
- End-users of this application who wish to read about what this project can do.
- End-users can use this document to read the capabilities of the application.

1.3 Product Scope

The scope of this document includes the following:

- The software products are **Anomaly Injection**, **Onboard Anomaly Detection**, and **Ground Base Anomaly Detection**. These products are further explained in 1.3.1, 1.3.2 and 1.3.3
- All the systems will utilize the Open Sat Kit environment. The software will use the simulated telemetry data and satellite data to inject and detect anomalies.

1.3.1 Anomaly Injection

Allows the user to inject an anomaly into the cFS. The anomaly injection will modify data that is onboard the satellite. It will also be able to interfere with the software bus.

1.3.2 Onboard Anomaly Detection

Automated software able to detect anomalies within the cFS. The onboard anomaly detection will make comparisons with the satellite data and nominal data

1.3.3 Ground Base Anomaly Detection

Automated software to be able to detect anomalies found in the telemetry data. The ground base anomaly detection will make comparisons with the telemetry data and nominal data

1.4 Definitions, Acronyms, and Abbreviations

1.4.1 Acronyms/Abbreviations

Acronyms/Abbreviations	Definition
API	Application Programming Interface
Attitude Det/Ctrl	Attitude Determination and Control Apps
cFE	Core Flight Executive
cFS	Core Flight System
DS	Data Storage
ES	Executive Services
EVS	Event Services
FM	File Manager
FWS	Flight Software
GBAD	Ground Base Anomaly Detection
GUI	Graphical User Interface
HK	House Keeping
LC	Limit Checker
OAD	Onboard Anomaly Detection
OSK	OpenSatKit
SB	Software Bus Services
SC	Stored Command
SimSat	Simple Satellite
TBL	Table Services
TFTP	Trivial File Transfer Protocol
TIME	Time Services

1.5 References

1.5.1 Aerospace detailed project proposal

1.5.2 OSK User's Guide

2. Overall Description

2.1 System Analysis

Main goals of this project:

- Inject anomalous behavior/data
- Use modern data analysis techniques/libraries/frameworks to detect those anomalies where traditional checks would fail

Major technical hurdles of this project:

- Detection of the anomaly either onboard the satellite and on the ground. The anomaly detection made onboard the satellite processor has limited memory and CPU cycles running on a single thread. The anomaly detection made for the ground has reduced response times due to data payload size when downlinking, as well as reduced opportunities to respond to the anomalies due to scheduling

2.2 Product Perspective

2.2.1 Anomaly Injection

The anomaly injection will be integrated into the cFS. The anomaly injection will make use of the cFE libraries to be able to connect with the other applications located on the cFS.

2.2.2 Onboard Anomaly Detection

The onboard anomaly detection will be integrated into the cFS. The onboard anomaly detection will make use of the cFE libraries in order to connect with the other applications located on cFS.

2.2.3 Ground Base Anomaly Detection

The ground base anomaly detection will be integrated into COSMOS. The ground base anomaly detection will make use of the cFE libraries to connect with COSMOS and the cFS.

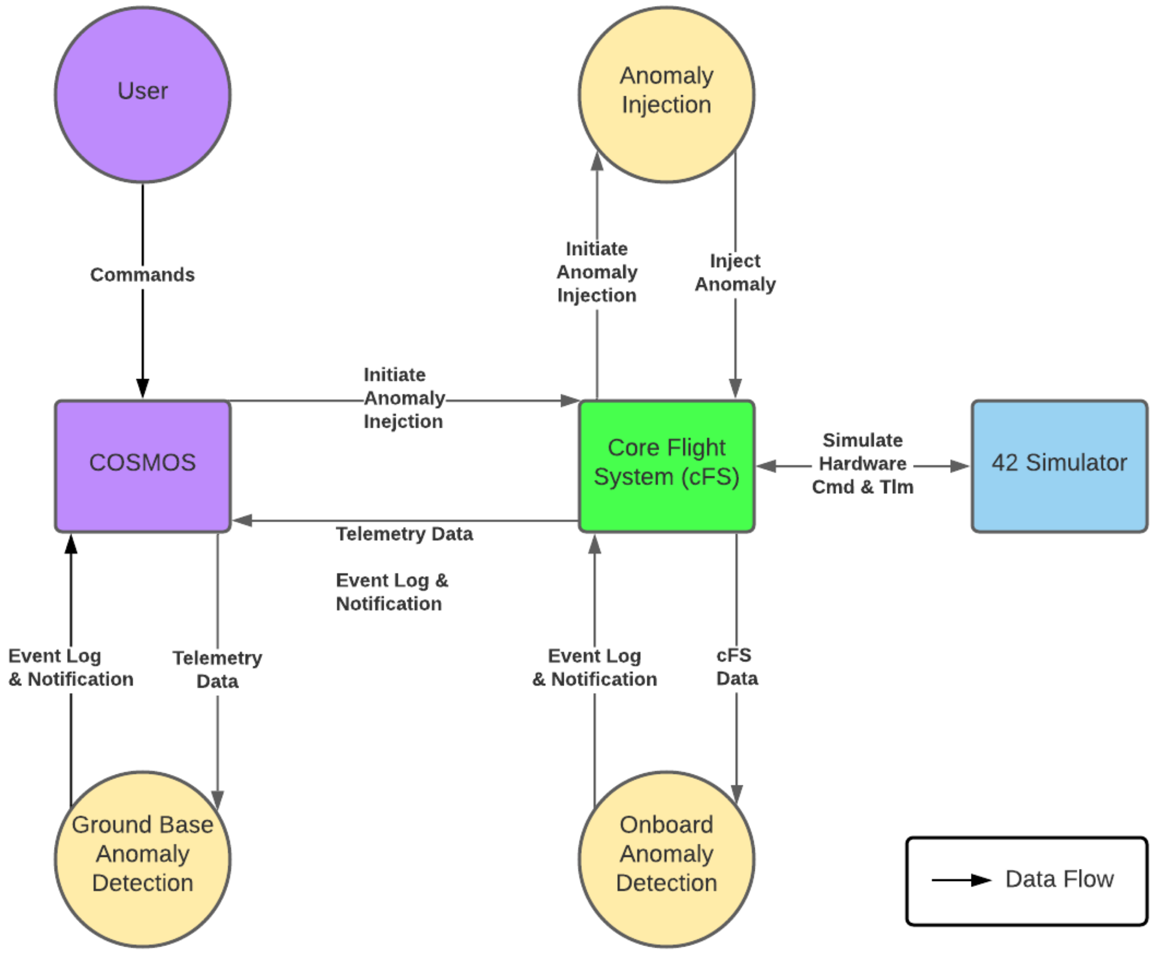

Figure 2-1 Product Data Flow Diagram

2.3 Product Functions

2.3.1 Anomaly Injection

- Injects the following anomalies:
    - Runaway task - Eats up the CPU of the cFS
    - Memory leak - Eats up the memory of the cFS
    - Denial of service - Occupies the bandwidth of the OSK softwarebus
    - Invalid command sequences - Affects the execution of commands in the cFS
    - Single bit errors - Affects the memory data of the cFS
- The injection software will allow the user to input their anomaly of choice
- The injection software will have an automated process that will wait until a period of time has passed before injecting one of the anomalies at random.
    - The user can turn this feature on and off.

2.3.2 Onboard Anomaly Detection
Automated detection software checks for anomalies. The anomalies being checked for are runaway tasks, memory leak, and single bit errors. Once an anomaly has been detected it creates and event log and notification of the anomaly. An event log and notification are queued and downlinked to the ground system next contact.

2.3.3 Ground Base Anomaly Detection
Automated detection software checks for anomalies from the telemetry data. The anomalies being checked for is the denial of service and invalid command sequences. Once an anomaly has been detected it creates an event log and a notification of the anomaly. The event log and notification are sent to ground.

Note: We understand Aerospace's interest in the development of a broad anomaly detection program that can use modern data analysis techniques, such as time series analyses and machine learning. However due to the team's limited knowledge of such techniques implementation is out of scope.

2.4 User Classes and Characteristics
The primary demographic for this software is Aerospace engineers looking into anomaly injections and detection software. The anomaly injection can be used for all simulation purposes. The anomaly detection can be studied and used for the actual satellites that will be launched.

2.5 Operating Environment
The software will run on the Ubuntu 18.04 LTS operating system, as OSK currently is supported on that specific Linux system. It is expected to have OSK, cFS, COSMOS, and 42 installed to make use of the desired open source software for development.

2.6 Design and Implementation Constraints
- On board satellite constraints of limited memory and CPU cycles running on a single thread
- Ground communication limitation of reduced response times and payload limitations

2.7 User Documentation
- Anomaly Injection User Manual in pdf form
- Anomaly Detection User Manual in pdf form

2.8 Assumptions and Dependencies
- Ubuntu Operating System (18.04 LTS)

2.9 Apportioning of Requirements

TBD

3. External Interface Requirements

3.1 User Interfaces

TBD

3.2 Hardware Interfaces

Not Applicable

3.3 Software Interfaces

Open Sat Kit (OSK) – Core Flight System Starter Kit

Open Sat Kit combines three powerful open source tools that are currently used in real missions today: Ball Aerospace Corporation's COSMOS ground system, NASA Goddard's core Flight System(cFS) flight software, and NASA Goddard's 42 satellite simulator. See the documentation for OSK in its GitHub Wiki.

Core Flight System – Flight Software

OSK provides a complete desktop solution for learning how to use NASA's open source flight software (FSW) platform called the core Flight System (cFS). The cFS is a reusable FSW architecture that provides a portable and extendable platform with a product line deployment model. The cFS has significant flight heritage, provides a complete set of command and data handling functions required by most spacecraft, and is reliable. A virtual environment with OSK set up will serve as the development environment for learning about flight and ground system communications as well as providing a platform for developing anomalies to be injected into the simulation. OSK comes with the cFS preconfigured for a fictitious satellite called SimpleSat (SimSat).

42 – Spacecraft Simulator

In addition to cFS, OSK uses NASA Goddard's 42 dynamic satellite simulator for simulated hardware command and telemetry. 42 is a comprehensive general-purpose simulation of spacecraft attitude and orbit dynamics. Its primary purpose is to support design and validation of attitude control systems. 42 accurately models multi-body spacecraft attitude dynamics as well as modelling environments from low Earth orbit to throughout the solar system. It also features visualization of spacecraft attitude.

COSMOS – Ground System

OSK implements extensive COSMOS configurations and customizations so COSMOS can serve as the primary OSK user interface. COSMOS is a suite of applications that can be used to communicate with the satellite, monitor its performance and health, and display its data. The systems that COSMOS interfaces with can be anything from test equipment (power supplies, oscilloscopes, switched power strips, UPS devices, etc.), to development boards (Arduinos, Raspberry Pi, Beaglebone, etc.), to satellites.

COSMOS implements a client server architecture with the Command and Telemetry Server and the various other tools typically acting as clients to retrieve data. The Command and Telemetry Server connects to the targets and sends commands and receives telemetry (status data) from them. Targets are the items you are trying to control or get status from.

3.4 Communications Interfaces

TBD

(We are aware that this is intended to be a multiple system project, but we don't as of yet know how to go about this.)

4. Requirements Specification

4.1 Functional Requirements

Requirement No.	Requirement Description
4.1.1	**Anomaly Injection**
4.1.1.1	The anomaly injection shall inject the following anomalies ● Runaway Tasks ● Memory leaks ● Denial of Service ● Invalid Command Sequence ● Single Bit Errors *Rationale: These are the anomalies that was proposed to be injected*
4.1.1.1.1	The runaway task shall create a task in the cFS that does not terminate *Rationale: A runaway task is when a task does not terminate appropriately on the satellite. To replicate the anomaly, the runaway task will create a task that never completes on the cFS*
4.1.1.1.2	The memory leak shall allocate memory on the cFS without deallocating *Rationale: Memory Leak is when memory is allocated but is not appropriately deallocated on the satellite. To replicate the anomaly, memory will be allocated without deallocating on the cFS.*
4.1.1.1.3	The denial of service shall spam messages to the software bus *Rationale: Denial of service overwhelms the network of communication by spamming data packets into the network. In the OSK environment, the software bus is the network, and the data packets are called messages. To replicate the anomaly messages will be spammed into the software bus*
4.1.1.1.4	The invalid command sequence shall send a sequence of commands out of order *Rationale: Invalid Command Sequence is a sequence of commands that is sent out of order. To replicate the anomaly a sequence of commands will be sent out of order*

4.1.1.1.5	The single bit error shall flip a bit in the rewritable memory of the cFS *Rationale: A single bit error flips a bit of memory inside the satellite. Realistically the bit flip can occur anywhere in the memory, potentially crashing the whole system. Due to that concern, the injection will be targeting the rewritable memory of the cFS*
4.1.1.2	The anomaly injection shall have an automated process that randomly injects an anomaly at random times *Rationale: Anomalies in real life occur at any point in time. Creating an automated injection process will be able to replicate a real-life scenario of these anomalies.*
4.1.1.2.1	The automated process shall allow the user to turn it on and off *Rationale: For testing purposes, the user will need control when the automated process is turned on or off.*
4.1.1.3	The anomaly injection shall allow the user to inject their anomaly of choice *Rationale: For testing purposes the user will need control of what anomaly is to be injected.*
4.1.2	**Onboard Anomaly Detection (OAD)**
4.1.2.1	The OAD shall be an automated process that periodically check the state of the satellite. *Rationale: Due to the onboard CPU's limitations, the OAD will need to periodically check for anomalies instead of constantly checking.*
4.1.2.2	The OAD shall detect the following anomalies: ● Runaway Task ● Memory Leak ● Single Bit Errors *Rationale: Runaway Task, Memory Leak, and Single Bit Errors are anomalies found on the satellite themselves. Onboard detection will streamline the detection process.*
4.1.2.2.1	The runaway task detection shall acquire the CPU data from the cFS *Rationale: Runaway Task eats up the CPU resources of the satellite. The detection will need the CPU to determine if there is an anomaly.*

4.1.2.2.1.1	The runaway task detection shall detect anomalous behavior in the CPU data *Note: The detection process is still TBD. More work and research are needed to state what "anomalous behaviors" are.*
4.1.2.2.2	The memory leak detection shall acquire the memory state from the cFS *Rationale: Memory Leak eats up the memory of the cFS. The detection will need the memory state to determine if there is an anomaly*
4.1.2.2.2.1	The memory leak detection shall detect anomalous behaviors in the memory state *Note: The detection process is still TBD. More work and research are needed to state what "anomalous behaviors" are.*
4.1.2.2.3	The single bit error shall acquire the memory data from the cFS *Rationale: Single Bit Error will happen within the memory of the cFS. The memory data is needed to determine if there is an anomaly*
4.1.2.2.3.1	The single bit error shall detect anomalous behaviors in the memory data *Note: The detection process is still TBD. More work and research are needed to state what "anomalous behaviors" are.*
4.1.2.2	The OAD shall create an event log once an anomaly has been detected *Rationale: The anomaly needs to be recorded. An event log will create that recording*
4.1.2.3	The OAD shall create a notification once an anomaly has been detected *Rationale: The user needs to be notified of the anomaly. The notification will notify the user.*
4.1.2.4	The OAD shall send the event log and notification next contact with ground *Rationale: The ground users will need to know what happened on the satellite. Therefore, the notification and event log will be required to be sent back.*
4.1.3	**Ground Base Anomaly Detection (GBAD)**

4.1.3.1	The GBAD shall be an automated process that will constantly check the state of the telemetry data. *Rationale: Due to unlimited resources provided on the ground, the automated process can run constantly*
4.1.3.1	The GBAD shall detect the following anomalies: ● Denial of Service ● Invalid Command Sequences *Rationale: Denial of Service is an anomaly that slows down the connection time between the satellite and the ground. Since the users are on the ground, detection should be there as well. Invalid Command Sequences are commands that are sent out of order. Detecting this anomaly on the ground would prevent these commands from being sent up in the first place.*
4.1.3.1.1	The denial of service detection shall acquire the connection time between COSMOS and the cFS *Rationale: Denial of Service slows down the connection between satellite, ground, and vice versa. COSMOS and cFS are ground and cFS, respectively. The connection time is needed to determine if there is an anomaly.*
4.1.3.1.1.1	The denial of service detection shall detect anomalous behaviors in the connection time *Note: The detection process is still TBD. More work and research are needed to state what "anomalous behaviors" are.*
4.1.3.1.2	The invalid command sequence detection shall acquire the command sequence being sent up to the cFS *Rationale: Invalid Command Sequence is a series of commands that are sent out of order. Inspection on these commands would determine if there is an anomaly*
4.1.3.1.2.1	The invalid command sequence detection shall detect anomalous behaviors in the command sequence *Note: The detection process is still TBD. More work and research are needed to state what "anomalous behaviors" are.*
4.1.3.2	The GBAD shall create an event log once an anomaly has been detected

	Rationale: The anomaly needs to be recorded. An event log will create that recording
4.1.3.3	The GBAD shall create a notification once an anomaly has been detected *Rationale: The user needs to be notified of the anomaly. The notification will notify the user.*
4.1.3.4	The GBAD shall send an event log and notification back to ground *Rationale: The ground users will need to know if there was an anomaly. Therefore, the notification and event log will be required to be sent to the ground users.*

4.2 External Interface Requirements

Not Applicable

4.3 Logical Database Requirements

TBD (We currently don't know if we will end up needing one)

4.4 Design Constraints

Design No.	**Design Description**
1.	**Onboard Anomaly Detection**
1.1	Constrain of the limited CPU cycles and processor running on a single thread.

5. Other Nonfunctional Requirements

5.1 Safety Requirements

The most significant safety issue in mind is ensuring software accepts only valid templates for anomaly simulation, guaranteeing a way to cease software operation safely in case of erroneous use.
Ensure simulation of certain anomalies, such as runaway tasks, do not cause issues on the platform where software is hosted.

5.2 Security Requirements

Externally, the software intends to be open-source software and therefore requires little to no actual security. However, within the simulation desired, some form of authentication system such as a login system should be applied to facilitate "secure" information transfer between the simulation satellite and its ground side base and mitigate unauthorized access. Any other security requirements are TBD, as vulnerabilities are detected.

5.3 Software Quality Attributes

Most significant issue: software must run on Ubuntu 18.04 LTS operating system; key software components from OSK only support that OS. Hardware limitations should be exceptionally low, allowing for installation and operation on most platforms to support Ubuntu.

5.4 Business Rules

Once operational, the satellite simulation should be left mainly to run normally. The only time it does not run normally is when an anomaly has been injected to test for the detection capabilities. The ground base should be the primary user interface to work with the satellite within the context of the simulation itself.

6. Legal and Ethical Considerations

There are legal and ethical concerns that pertain to this project—the most significant being the anomaly injection tool being created. As stated by Aerospace, the anomaly injection tool will be breaking regulatory protocols that will lead to some indifference in the system. This change goes against the ACM Code of Ethics Section 1.2: Avoid Harm. We are creating the potential of unjustified property damage, so it will be addressed what kind of damages our program entails, how the developers and sponsors responded to the damages, and how adequate the responses are.

The anomaly injection tool has five possible anomaly injections. Those are the runaway task, memory leak, denial of service, invalid command sequences, and single bit errors. These anomalies all break the protocol and cripples the effectiveness of the satellite. Memory leaks and runaway tasks cripples the systems overall performance eating up the CPU and memory of the satellite respectively. Single bit errors change any bit of memory in the satellite, which could cause significant damages, including crashing. Denial of service causes disruptions in the data flow. Invalid command sequences disrupt command execution.

Safeguards will be implemented to address the issues. First of all, the anomalies are only injected by the user's knowledge. The user has control of when to inject these anomalies. There is also a development of an automated system to inject these anomalies as well. The user also controls this automated system in terms of an on and off switch. Any abnormality injected will be addressed to the user immediately. Second of all, a failsafe method will revert the satellite to its original functioning state, including deleting the previously injected anomalies. Third of all, the anomaly single bit error has its protocol. The protocol is only to target the rewritable memory of the satellite, therefore mitigating the potential crashes.

The efforts the developers are taking to address these issues are adequate. Letting the user know what anomalies have been injected at least states what kind of potential damage they are about to do. And addressing the single-bit error to be assigned to rewritable memory allows for ease of mind to not crash the system. The failsafe of reverting the satellite into its original state mitigates all damages done towards the satellite. These are adequate attempts to combat the potential harms. However, there can be improvements. The biggest one is to have some protocol for all anomalies, not just single-bit errors.

The ethical issue of this project is breaking the ACM Code of Ethics Section 1.2: Avoid Harm. The potential unjustified damages being done are the anomalies that damage the system integrity. These damages were addressed in user knowledge and consent, creating a failsafe, and managing the single-bit error. The responses to these issues are adequate, but there is room for improvement. Given the heavy emphasis on anomaly injection, it is essential to note the development of anomaly detection. There cannot be a detection tool without an injection tool. In the long run, we can address and prevent damages in the future by doing damages now. That is the primary goal of the anomaly detection tool upholding the ACM Code of Ethics Section 1.2: Avoid Harm.

Appendix A: Glossary

TBD

Appendix B: Analysis Models

Internal problems	**Manifestations**	**Plausible causes**	**Recognition strategies**
Denial of Service	The time for connection between satellite to ground or vice versa, is slower than scheduled	Jamming from an outside source	Checking scheduled time for connection vs current time of connection
Invalid command sequences	The satellite not executing the appropriate behaviors	Ground system not sending the correct order of commands. Attacker modifying the data to input an invalid sequence	Checking command sequence with a valid command sequence
Runaway tasks	The computer found on the satellite starts to run slowly	A process still existing after being cancelled	Checking the CPU performance of the satellite
Memory leak	The memory of the satellite is being eaten up	Memory has not been deallocated appropriately	Checking the memory availability
Single Bit Error	Discrepancies in the memory data	Solar radiation flipped the bit	Compare old state of memory to the current state of memory and check for changes