Proceedings of the Canadian Society for Mechanical Engineering International Congress 2020
CSME Congress 2020
June 21-24, 2020, Charlottetown, PE, Canada

Development of Automated Testing Infrastructure for a CubeSat On-Board Computer

1st Andrada Zoltan
Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
andrada.zoltan@alumni.ubc.ca

2nd Richard Arthurs
Mechatronic Systems Engineering
Simon Fraser University
Surrey, Canada
rarthurs@sfu.ca

***Abstract*—All spacecraft systems require thorough verification prior to launch, due to the complex interactions between subsystems and the inability to repair hardware once the system is in space. The verification process is often constrained by project timelines and budget, which is especially the case with CubeSat programs, where development time may span only a few years and development costs are kept to a minimum. These constraints sometimes lead to a lowered emphasis on rigorous testing, inducing technical risks into the project due to lack of proper verification. Existing literature on functional testing of CubeSats focuses mainly on ADCS performance verification and does not account for exhaustive system verification of the CubeSat as a whole. There is a need for inexpensive testing infrastructure that can be developed alongside the spacecraft itself, without impeding the progress of the project. The ORCASat automated testing system provides this capability.**

This system is designed to enable fully automated test sequences during all phases of development and throughout AI&T. It is developed alongside the spacecraft systems from an early stage and consists of a supplementary hardware platform and an application running on a host computer. The hardware platform enables simulation of adjacent subsystems in the satellite, which allows firmware development and testing to advance without completed hardware in place. Testability is also designed into the flight hardware and software, resulting in reuse of features and reducing duplication of work. The development timeline of such a system has been appropriately curated to match common milestones in a CubeSat project, and is presented here. As well, the design details of this testing system and recommendations are discussed, alongside expectations of how they will reduce testing time and costs and enhance system reliability.

***Index Terms*—automated testing, verification, on-board computer, CubeSat**

I. Introduction

There has been a rapid increase in the number of CubeSats launched over the past couple of years. Many companies and universities choose CubeSat development due to the faster timeline and less expensive costs when compared to larger satellite platforms [1]. Testing of on-board computer hardware and software for CubeSats is often weak or insufficient due to the lack of time and money. As a result, many CubeSat programs end in mission failure [2] [3].

The ORCASat project is a Canadian Space Agency funded project, involving the development of a 2U CubeSat used for optical telescope calibration. The Command and Data Handling (C&DH) system consists of a custom on-board computer (OBC) design made from commercial-off-the-shelf components and an in-house firmware architecture built upon FreeRTOS. The use of a custom, and not already space-qualified OBC, brings about the need for an interface-compatible, bespoke testing solution. Full verification of the system must be executed in-house.

This paper describes the testing framework developed for the ORCASat OBC, outlines the methodology used to design this system, and explains the design choices that were made in detail. The purpose of this paper is to show how the development of an automated testing framework can be tightly integrated into the overall development of the OBC subsystem, in order to ensure that the OBC can be thoroughly tested before flight.

II. Background

A. Hardware-in-the-Loop Testing

Hardware-in-the-loop (HIL) testing refers to the practice of executing tests on hardware with the addition of simulated inputs, and monitored outputs. It is a technique used for low-cost, thorough, and repeatable testing of embedded systems [4]. In the context of CubeSats, HIL testing is typically performed in the later stages of spacecraft integration [5]. Sensors and actuators that are difficult to verify in a laboratory environment are simulated as inputs to the system, and the system's responses and behaviour are monitored. Although HIL testing is largely useful for this purpose, its benefits can be reaped in earlier stages of the project for testing of the OBC.

Development of the ORCASat OBC occurs without any physical connection to external systems for the first stages of the project, leading to a large quantity of firmware being developed without full verification. Due to the tight coupling between hardware and firmware, issues may not be uncovered until much later into the project where integration begins. However, the introduction of HIL testing early on can reduce this risk and allows for continuous testing of the OBC as the system matures. This practice can also reduce the time for testing during the AI&T phase, as the fundamental firmware issues have already been discovered and resolved.

B. Test as You Fly

"Test as You Fly" or "Test Like You Fly" (TLYF) is a philosophy in spacecraft engineering that revolves around the preference for including flight-like hardware, software, procedures, and other mission components in the testing program of the spacecraft [6]. With a TLYF approach, the satellite is tested with flight-like timing and commands, using as much flight infrastructure as possible. This helps verify that the actual mission can be executed end-to-end, and that on-orbit operations will not be the first time that the spacecraft experiences certain commands and inputs. The downside of TLYF testing is that certain tests may take a long time, such as those involving flight-like telemetry collection rates. For this reason, many spacecraft undergo a hybrid approach that involves TLYF, but also includes increased cadence tests to hit increased levels of coverage in smaller amounts of time [7].

C. Related Work

Recaps of CubeSat programs typically stress the need for more testing of all types [3]. On larger spacecraft, the "test as you fly" approach is a standard methodology applied on many missions, but the need for increased automation and measurement of system parameters has been noted as an improvement to incorporate in future missions [7]. The Small Projects Rapid Integration and Test Environment (SPRITE) system from Marshall Space Flight Centre is an example of a flexible testing system that includes power measurement features, and has been applied to small satellite hardware. SPRITE was used to verify a Guidance, Navigation, and Control system for a 3U satellite, and is capable of supporting test activities for other satellite systems with its extensive simulation capabilities [8].

Despite the recognized need for more thorough testing, many CubeSat testing programs focus on tests at the end of the development phase, with all subsystems integrated [9] [5] [1]. As a result, problems in individual subsystems are often caught late in the development cycle [9]. However, some literature about CubeSat software development does mention testing during subsystem development [10] [11].

III. Architecture

A. System Overview

The ORCASat C&DH testing system consists of three major components: the design under test or OBC, a HIL test platform (HTP), and a desktop application written in Python known as Houston. The three components are closely integrated across hardware and software interfaces, forming the full system seen in Figure 1. The HTP is an auxiliary circuit board that connects to the OBC over the satellite bus and a debug connector. The satellite bus is used to exercise the OBC's electrical interfaces to the rest of the satellite, and the debug connector is used only for testing. The Houston application provides a simple method to interface with both the OBC and the HIL test platform over a serial interface. Houston allows for graphical command creation, monitoring of telemetry, and execution of tests.

The infrastructure around the OBC was developed with the intention of adopting a test-as-you-fly approach, allowing the OBC firmware and hardware to maintain as close to its flight configuration as possible. The addition of the HIL test platform was a result of needing to non-intrusively monitor signals on the OBC, as well as physically interface with the OBC's communication peripherals for hardware verification. By introducing this secondary circuit board into the system, all debug hardware components may be taken off the OBC, leaving it with only the necessary hardware for flight, even at a prototype stage of development.

On the software side, the interface between all three components takes the form of commands, responses, and telemetry files. By employing a similar communication interface as to that used during flight, the OBC's command system may be exercised extensively throughout development. This not only exhaustively verifies the on-board command system, but also tests the command scheduling and uplinking capabilities of Houston. Although Houston is used for testing during development, it will later be employed as the mission control software on the ground during flight, so it is important to verify its abilities early.

B. HIL Test Platform Capabilities

The HIL test platform interfaces with the OBC over the satellite bus for signal monitoring and subsystem simulation. Signal monitoring capabilities are used continuously during most tests of the OBC. Subsystem simulation capabilities are added to the HTP firmware on an as-needed basis, depending on the testing requirements and development level of external subsystems. Since the HTP has complete access to all satellite bus signals, simulation of external subsystems is done primarily with firmware. Simulating with firmware means that failure cases that are not possible to simulate with real subsystem hardware can be implemented, which is a key benefit of the HTP design. The ability to simulate subsystems flexibly in firmware is also beneficial when developing with COTS subsystems, which have long lead times and may not be easily available to OBC developers if only one flight unit exists.

The HTP also has control over the reset state of all hardware components on the C&DH board through the debug connector. Reset control allows for simulation of complete on-board component failure during testing, allowing the OBC's fault-tolerant firmware capabilities to be exercised. Simulation of on-board component failure could also be done within OBC firmware itself, but would require the addition of non flight-like capabilities in order to virtually turn off components. External reset control of these components allows unmodified OBC firmware to be used for this sort of testing.

C. Houston Capabilities

Houston is a generic platform designed to support testing and mission operations. OBC developers can construct and send commands, and view live telemetry streams from a connected OBC during development. Houston can also execute

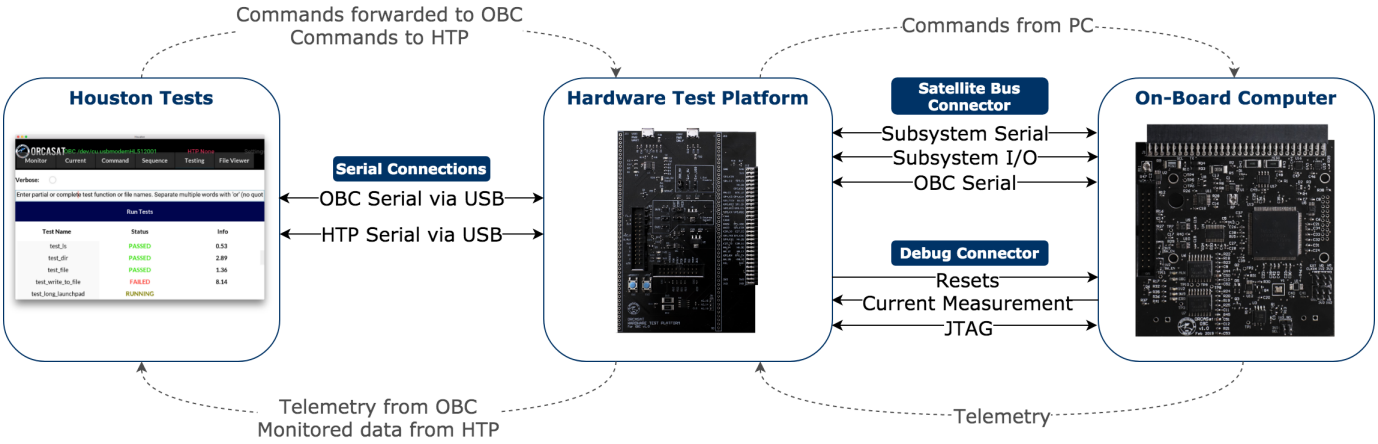

Figure 1. ORCASat C&DH Testing System Components

automated test sequences. Much of the functionality required for testing the OBC, such as command scheduling and parsing of telemetry data, is also required for mission operation. The infrastructure developed first for OBC testing is designed to be reusable during flight, reducing development duplication, and ensuring that by design, the OBC is tested as it will be flown.

D. Automated Testing

The ORCASat automated testing system was developed with ease of use and low overhead in mind. At its core, an automated test is driven by commands sent through Houston and is verified by checking the returned log messages and telemetry from both the OBC and HTP. Tests send ASCII commands to control the OBC and schedule it to perform tasks, as would be done in flight. For HTP control, commands are used to invoke monitoring functionality or to trigger the HTP to simulate a faulty input going to the OBC. Stimulus generation can occur rapidly, or at a flight-like cadence, controllable from within the test run by Houston. The OBC's response to the stimulus can then be checked using the reported logs.

Tests are written in Python, and are run using the Pytest framework. Test status and results are passed to the Houston user interface to be displayed. Test files interact with boards using a series of Python classes that were written to send commands to either board and check responses. The use of a high-level language has made the development of complex tests and new testing infrastructure very straightforward. For example, tests that schedule OBC commands at random times can be implemented with standard Python features such as random number generation. Using a high-level language also allows for intricate checking of data returned from the OBC to be written easily, reducing the overhead required when implementing tests.

Once tests are written, they are version controlled using git. The suite of tests is updated as new firmware features are added, and expecting that all tests pass is an important checklist item in the firmware development process. The ability to easily run and rerun tests allows regression testing throughout the development of the OBC and its firmware.

E. Data Handling in Tests

Two sources of telemetry data are available from the OBC during tests. The first is the "live" telemetry stream that consists of informational log messages, which are transmitted immediately by the OBC in response to commands, anomalies, or other important events. The live stream of telemetry is very useful during development and early testing, where immediate and verbose responses are useful. However, it is not flight-like because during flight, ORCASat will only transmit data upon request.

The other source of data is telemetry data files, which contain many of the same messages as the live stream. Certain data files also contain periodically-collected telemetry from all subsystems of the satellite. Data saved to files is stored on the OBC until a downlink request is received. On-orbit, the OBC will not be transmitting the live telemetry stream due to radio licensing and power consumption considerations. All data that will be downlinked is saved to a file, and is downlinked by command when ORCASat is in range of a ground station.

During most early tests, the live stream is monitored for desired and undesired responses to particular commands and events. The consistent format of the messages in the live stream makes it straightforward to define and check the expected behaviour of the satellite in test code, and therefore, classify if a test passes or fails.

During flight-like tests, the telemetry data files are used as a source of verification instead. Once the OBC's responses are to be examined, Houston transmits a file downlink request and acquires the on-board files using the same protocol used in flight. The contents of the file are examined automatically based on the expected message types and timestamps required by the test. Different files on the OBC take on different formats, but for functional testing of the OBC, the primary files of concern are the system and error logs, which are populated with messages identical in format to those from the live telemetry stream. Files for external subsystem teleme-

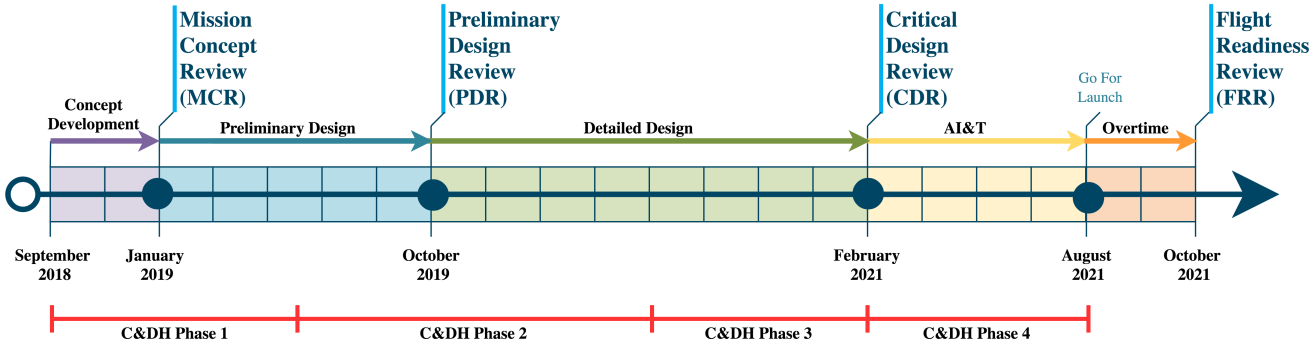

Figure 2. C&DH Phases Timeline

try typically have timestamped measurements of engineering parameters such as voltage, temperature, or current. These parameters are expected to be checked during later phases of testing when the OBC is connected to real subsystem hardware.

IV. Methodology

A. Development and Test Phases

In a CubeSat project, time and budget are limited resources that drive a significant portion of the work that is completed. For this reason, developing an automated testing framework is typically not a high priority. The framework outlined in this paper is intended to be a low-cost solution to this problem, that provides great benefits during development, and results in creation of components that are required for running the mission. By splitting the development of this infrastructure into different phases, closely following the development of the OBC itself, the overhead incurred by developing test infrastructure can be minimized.

The work is split into four phases, scheduled as seen in Figure 2:

- **Phase 1 - Initial Bring-up:** The purpose of this phase is to test the basic hardware functionality of the OBC, verifying both the firmware drivers and the hardware devices. In this stage, in-firmware tests are used on the OBC to verify some core functionality of the hardware. In this first phase of development, Houston allows operators to visually confirm that the board is operating as expected, by providing a user interface for displaying live log messages. The Houston Pytest infrastructure is not developed in this phase, to allow for focus to be placed on developing and verifying the core features of the OBC, which is important in the early phases of the project to prove the feasibility of the C&DH system as soon as possible.
- **Phase 2 - System Software:** The second phase of development begins once all the hardware has been evaluated for basic functionality and is working as expected. At this point, the OBC firmware consists of core infrastructure and low-level drivers. Firmware moves onto the development of key system features, such as the error handler and command scheduler. In this stage, the HTP subsystem is provided with monitoring functionality for some of the physical OBC features, such as the watchdog heartbeat, on-board GPIO expander, and current draw. Houston also gains the functionality to actively control the OBC through the use of automated tests, and downlink log files from the OBC. During this phase, automated testing because a regular piece in the development process. No features may be added to the OBC main code base until all tests are passing, and new tests are added weekly to support the verification of new features. The purpose of this second phase of testing is to verify the internal features of the OBC prior to moving on to subsystem integration. This phase enables continuous testing of the OBC as new features are added, and ensures that issues are caught early and often.
- **Phase 3 - Subsystem Integration:** The purpose of this phase is to prepare the OBC for Assembly, Integration & Testing (AI&T). In this stage, the OBC has all telemetry, command and error handling interfaces with other subsystems in place. To verify these interfaces prior to AI&T, HTP functionality is expanded to allow for mocking of the relevant subsystems. Error injection is also added to the HTP's capabilities, to help verify corner cases that may not otherwise be possible with the real hardware connected. By performing this testing phase prior to AI&T, a majority of the errors in the OBC firmware can be uncovered, allowing the AI&T process to follow smoothly afterwards.
- **Phase 4 - Mission Life-cycle:** The final phase of development comes with the completion of the OBC hardware and firmware. The project as a whole is in the AI&T phase, where testing of the entire satellite begins. The existing test sequences are executed to verify functionality of the OBC while it is integrated with the rest of the satellite. Additionally, increased cadence and day-in-the-life tests are added, with focus on the testing needs of non-OBC subsystems. These tests are all still run

by Houston interacting with the OBC, and tests may be executed for qualification before and after satellite-wide physical tests, such as vibration tests.

By breaking the development of the automated testing framework into phases that directly support the OBC's development timeline, new firmware features can be tested as soon as they are developed, and regressions can be caught from an early stage. The presence of this framework throughout all project phases allows for easier verification at each phase, while only developing the necessary test infrastructure to suit the current level of development. The end result is a system that can support fully automated test sequences in any configuration of HTP simulation and real subsystem hardware. Such a system allows for repeatable, consistent and controllable testing.

B. Designing the Testing Platform Hardware

The need for a HIL test platform was recognized early in the C&DH system design process to enable flexibility for how and when the OBC can be tested. HIL systems tend to rely on commercial hardware such as DAQs [12], or on low-cost hobbyist-level devices such as Arduino [13]. Both of these choices are likely made for integration concerns. Commercial tools come with software support for automation, and hobbyist-level tools are straightforward to use.

For ORCASat C&DH, the HIL test platform is based on a development board for the TMS570, the same series of medical-grade processors used in the OBC, and a custom PCB that mounts on top of the development board. This platform was chosen to reduce costs and to enable reusability of firmware components between the two boards. By using a low-cost development board as the computing platform and a simple 2-layer PCB, the test platform hardware costs are reduced greatly while useful functionality is maximized.

The custom PCB design began with an analysis of the interfaces present on the OBC, coupled with the planning of tests that the hardware must support. Additionally, features to support day-to-day OBC firmware development were added to the HTP, as the OBC requires power supplies and serial interface conversion. The following hardware features are present on the HTP:

- OBC reset buttons, which are useful during development.
- USB to UART converter, allowing the entire device to connect to a host PC using a single USB port (for OBC firmware development) or two USB ports (for full simulation capability).
- ADC channels allowing current monitoring of the OBC power supplies.
- Mapping of important OBC signals such as watchdog trigger pins to the HTP microcontroller, allowing them to be monitored.
- Flexible power supplies - the OBC and HTP stack can be powered from a USB port for portable development, or can be powered from bench power supplies for in-lab testing.

The HTP also includes a test harness called the "debug connector" which maintains the set of signals that are not required during flight, but are required to adequately test the system - whether it be verifying a particular design detail, or injecting an anomaly such that the detection of the anomaly can be verified from elsewhere in the system. Determining which signals should be made available for the test harness requires tests to be conceptualized on a component level during the hardware design phase. On the ORCASat OBC, the following signals were brought out on to a separate testing header to facilitate automated testing of fault scenarios.

- **Peripheral device power/reset:** Power and reset signals are important for testing total failure cases, and for isolating devices during system power measurements. If a peripheral can be reset or turned off completely, the firmware response to the complete failure of the device can be checked.
- **Current monitor outputs:** The ORCASat OBC features on-board current monitoring. The current sense amplifier outputs are broken out to the test header to allow constant external measurement of current.
- **Hardware alarms:** Signals that are hardware-driven but are separate from the OBC processor are difficult or impossible to check with OBC firmware. For this reason, they should be brought out to a header for testing purposes.

C. Monitoring OBC Signals

HTP firmware implements monitoring functionality using individual FreeRTOS tasks. For example, the task that monitors the OBC's watchdog heartbeat signal is started automatically by command at the beginning of a test. The task begins to run, and reports properties of the signal to Houston, such as its period. If the OBC fails to pet the watchdog during the test, the monitor task detects this, and sends an appropriate message to Houston. In most tests, detection of a watchdog petting failure would cause the test to fail, as it indicates hung firmware. However, flexibility is provided in some tests, such as those verifying that the OBC can reboot itself, the testing infrastructure can be configured to expect a certain failure indication from a monitor.

Other monitor tasks on the HTP include those for measuring OBC current and OBC general-purpose outputs. In most cases, the outputs of these monitors are checked automatically in the background of in every test, to catch issues such as spurious OBC outputs, or to record current consumption figures. Since each monitor task is a simple and self-contained element of HTP firmware, and their interface to Houston is consistent, adding continuous monitoring of OBC effects to all test scenarios has been simplified.

D. Test Planning

As prior mentioned, the infrastructure outlined in this paper is developed on an as-needed basis, which drives the timeline of each phase. To determine which features are needed, it is necessary to develop a test plan at the start of each phase or

at major milestones throughout the project. With a test plan in place, each test can then be assessed for required infrastructure needed to enable execution of the test. Defining work based on the test plan ensures that no unnecessary features are added, minimizing the overhead of maintaining this infrastructure.

The ORCASat test planning process builds off of the requirements of the system, and is driven by the features expected to be completed by the end of a phase. The requirements of each feature are taken and broken into test cases that verify the system's compliance to these requirements. Each test case is then converted into an automated test in Houston and added to the repertoire of tests used to verify the system. By the end of a phase, this process accomplishes verification of all added features in that phase, meeting the goal of continuous integration and testing of the system.

V. Conclusions and Recommendations

The ORCASat C&DH testing system has been successfully used to find bugs in OBC firmware during development, and to verify firmware functionality against requirements. Many of the firmware bugs found were introduced by new changes and would have gone unnoticed without testing, potentially manifesting later during AI&T, or even during flight. With the testing system at its current level of development, it is straightforward to add tests for new features, and regression testing can be done with a single button click. The following recommendations are worth considering by any CubeSat subsystem team who develops firmware or custom hardware:

- **Develop prototypes with testing in mind:** break out important signals, and consider which failure modes may not be possible to simulate in firmware.
- **Automate testing as much as possible:** once the initial hurdle of automating a single test is complete, the incremental effort required to automate future tests is likely minimal. This allows for effort to be placed on increasing the coverage of tests, as the time taken to introduce new tests is negligible.
- **Consider a high-level language for testing infrastructure:** high-level languages allow for fast creation of test infrastructure because they have feature-rich libraries and straightforward syntax. Third-party packages can be leveraged to build complex features quickly.
- **Plan testing alongside firmware development:** having test infrastructure ready when firmware is developed means new features can be tested thoroughly as they are written. Developing either firmware or tests well ahead of time risks wasting effort, as details may change in the future.
- **Test extensively at the system level before integrating:** integrating with multiple known-good systems helps isolate any problems to the act of integration itself. Ideally, most testing that can be done with a non-integrated system can also be done on the integrated system, ensuring thorough proof that no issues were introduced by integrating. Tests specific to the capabilities of the integrated system should also be performed, such as full day-in-the life mission simulations.

The ORCASat C&DH team continues to use and improve our testing infrastructure. We expect that with it, we will uncover and be able to fix more firmware bugs before AI&T, and deliver a robust system that will allow the mission to achieve its goals.

Acknowledgment

This work is supported by the generous funding of the Canadian Space Agency (CSA), as part of the Canadian CubeSat Project. Authors are grateful for the technical guidance, resources and time the CSA has donated to this project.

References

[1] I. Arslan, K. Arslankoz, A. Telli, B. Karabulut, and A. R. Aslan, "On-board software for havelsat cubesat," in *2017 8th International Conference on Recent Advances in Space Technologies (RAST)*, 2017, pp. 229–233.

[2] C. Venturini, B. Braun, and D. Hinkley, "Improving mission success of cubesats," 2017. [Online]. Available: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=4085&context=smallsat

[3] L. Berthoud, M. Swartwout, J. Cutler, D. Klumpar, J. A. Larsen, and J. D. Nielsen, "University cubesat project management for success," in *Proceedings of the 33rd Annual AIAA/USU Conference on Small Satellites*. Utah State University, 2019. [Online]. Available: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=4434&context=smallsat

[4] J. A. Ledin, *Hardware-in-the-Loop Simulation*, 1999, pp. 42–60. [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Embedded-Control-Systems/AdditionalMaterial/Applications/APP_Hardware-in-the-Loop_Simulation.pdf

[5] S. Corpino and F. Stesina, "Verification of a cubesat via hardware-in-the-loop simulation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 4, pp. 2807–2818, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6978879

[6] A. W. Bucher, "Test like you fly [spacecraft]," in *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, vol. 7, March 2001, pp. 7–3327 vol.7.

[7] S. Thibault, "A needed evolution in automated satellite testing," in *2007 IEEE Autotestcon*, Sep. 2007, pp. 425–434.

[8] A. Lee, J. Rackoczy, D. Heater, D. Sanders, and S. Tashakkor, "Small projects rapid integration and test environment (sprite) an innovation space for small projects design, development, integration, and test," in *16th Annual Space and Missile Defense Symposium*. Marshall Space Flight Centre, 2013. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140002971.pdf

[9] K. Rankin, I. McNeil, I. Rankin, and S. Stochaj, "How not to build a cubesat–lessons learned from developing and launching nmsu's first cubesat," 2019. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2019/all2019/54/

[10] S. F. Hishmeh, T. J. Doering, and J. E. Lumpp, "Design of flight software for the kysat cubesat bus," in *2009 IEEE Aerospace conference*, 2009, pp. 1–15.

[11] G. D. Manyak, "Fault tolerant and flexible cubesat software architecture," 2011. [Online]. Available: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1600&context=theses

[12] H. D. Nguyen and I. A. Miller, "Cost impact of automated acceptance testing of electrical ground support equipment for spacecraft testing," *IEEE Instrumentation Measurement Magazine*, vol. 15, no. 4, pp. 28–33, August 2012.

[13] W. Tapsawat, T. Sangpet, and S. Kuntanapreeda, "Development of a hardware-in-loop attitude control simulator for a cubesat satellite," *IOP Conference Series: Materials Science and Engineering*, vol. 297, p. 012010, Jan. 2018.