2022 IEEE/SICE International Symposium on System Integration (SII)
January 9-12, 2022. Narvik, Norway

2022 IEEE/SICE International Symposium on System Integration (SII) | 978-1-6654-4540-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/SII52469.2022.9708802

Testing of Software-Intensive Hyperspectral Imaging Payload for the HYPSO-1 CubeSat

Sivert Bakken[1], Roger Birkeland[2], Joseph L. Garrett[1], P. Amund R. Marton[1], Milica Orlandić[2], Evelyn Honoré-Livermore[2], Dennis D. Langer[3], Cecilia Haskins[4], and Tor A. Johansen[1]

***Abstract*— The growing community of CubeSats vendors makes it possible to launch and fly novel payloads for targeted applications by procuring flight-proven CubeSat platforms. The importance of embedded software for such Commercial Off-The-Shelf (COTS) based payload systems has increased to provide more functionality and flexibility. As COTS components are not designed for space, they warrant extensive software and hardware testing. The ongoing work on how the payload software testing procedure is used in the development of the HYPerspectral Smallsat for Ocean observation (HYPSO-1) satellite is presented. This paper discusses the strategy of software development, the challenges that were encountered, and the lessons learned throughout the process. In particular, the advantages of rehearsals, reviews, and manual testing are compared to automated and programmer-driven testing.**

I. Introduction

In this paper, we present the experiences and findings from software system integration testing of a hyperspectral imager (HSI) payload for a university CubeSat mission. A picture of the CubeSat going to space is given in Fig. 2. The HYPSO-1 spacecraft is primarily a science-oriented technology demonstrator enabling low-cost and high-performance hyperspectral remote sensing and autonomous onboard processing to collect ocean color data products in collaboration with other autonomous platforms [1], [2]. The onboard processing is intended to provide relevant information from and to other assets.

The HSI collects light reflectance spectra in the visible to the near-infrared wavelengths. These wavelengths are observable at the ocean surface and used to monitor important biomarkers, which can be used for several applications [1].

The HYPSO-1 satellite will be the first scientific small satellite developed at the Norwegian University of Science and Technology (NTNU), with a launch planned for Q4 2021. The payload is an in-house developed, low-cost HSI, mostly based on COTS components [3]. Functional and integration tests using breadboards and an Engineering Model (EM), Qualification Model (QM), and Flight Model (FM) model philosophy have been carried out [4]. Fig. 1 shows the relevant project progression during the final part of the payload development and integration, and the most relevant test types at different stages. The team is now working on its second satellite, the HYPSO-2, which will be launched in 2024.

Contribution

Here we present how the HYPSO-1 team performed the testing of the payload software, and how testing strategies were adapted to support our development and integration challenges. These adaptions made the payload software testing feasible, and possibly adequate, for the size of the team and the mission's acceptable risk. The testing focused on the early use of target hardware and enabled early integration [5]. Rehearsals, reviews, and manual testing are compared to automated and programmer-driven testing for our system.

Section II presents the background and related work. The testing strategies are described in section III. The results from the testing process are described in section IV. A discussion of the findings and lessons learned are given in section V. Lastly, in section VI, we provide conclusions.

II. Background and Related Work

The CubeSat standard is a simple small satellite reference model [6]. This standard is intended for low-cost satellites with a short project cycle relative to traditional space missions, both in terms of development, launch, and operations [7]. A CubeSat is based on the form factor of "cubes" or "units" with each edge measuring 10 centimeters. These units can be combined to create satellites with different form factors, e.g., 1U, 3U, 6U or larger [7]. With this mechanical envelope standardization, it is possible to launch several CubeSats from a single launch vehicle, and there are multiple CubeSat spacecraft and subsystem vendors available. Due to the standardization of the mechanical interfaces a CubeSat team can base their choice of vendor freely.

In this paper, the focus is on the development and testing of the HSI payload as an onboard processing platform, as well as its integration with the rest of the satellite bus provided by the CubeSat vendor. The *payload processor* or Onboard Processing Unit (OPU) receives and transmits commands and telemetry, as well as raw or processed payload data, to other satellite bus subsystems. The OPU does not play a direct role in spacecraft telemetry and telecommand data management. Those subsystems are provided by the CubeSat vendor. This simplifies the design of the CubeSat

*This work is supported by the Norwegian Research Council (grant no. 270959), the Centre of Autonomous Marine Operations and Systems (NTNU AMOS, grant no.223254), the Norwegian Space Center, the European Space Agency (PRODEX - 4000132515), and NO Grants 2014 – 2021, under project ELO-Hyp, contract no. 24/2020.

[1] The Norwegian University of Science and Technology, Department of Engineering Cybernetics
[2] NTNU, Department of Electronic Systems
[3] NTNU, Department of Marine Technology
[4] NTNU, Department of Mechanical and Industrial Engineering
Corresponding Author: sivert.bakken@ntnu.no

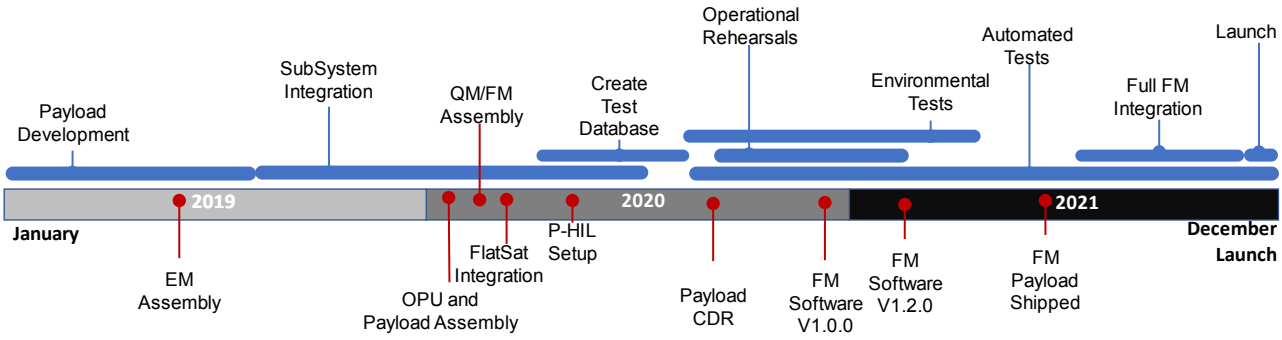

Fig. 1: Relevant development and testing timeline for HYPSO-1

and provides more resources in the development of the payload [7].

The European Cooperation for Space Standardization (ECSS) body of standards encourages reviews to assure better documentation, and states it as an opportunity for stakeholders to get an overview of the system details [6], [8]. Quality is assured in space missions by *reviews* and *tests* [5], [6]. Tests should demonstrate that the implementation meets the functional and performance requirements. CubeSat projects often discover unexpected and undesirable behavior during testing and integration, and the importance of these activities are strongly emphasized [4], [5].

In a university setting, there are challenges related to team organization; such as availability of (qualified) students at the right time as well as a high turnover [4], [5], [9]. This influences the whole development process; from design, implementation, test, and delivery. For many university teams, it is difficult to successfully transfer knowledge to new team members. Through trial-and-error, the HYPSO-1 team adopted a version of an agile digital engineering workflow described in [9]. The design of the software architecture to accommodate these challenges is not covered here.

To mitigate the lack of a dedicated test team, all team members are encouraged to contribute to testing. This makes it even more important to adapt and implement clear and efficient development and test routines.

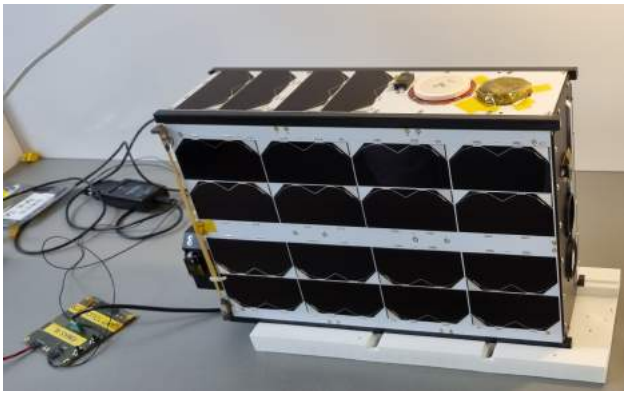

Fig. 2: An image of the 6U CubeSat going to space during final system check at the CubeSat vendors facilities.

Complex On-Board Data Handling systems are typical for technology demonstrating CubeSats, where more services are performed onboard. This requires more software to be developed to interface, control, and operate different subsystems, which in turn leads to a need for more testing activities.

In the literature, several approaches for software testing and integration activities of CubeSats have been reported. The focus is often on Software In the Loop (SIL) [5], [6], [10], Model In the Loop (MIL) [4], [6], and/or Hardware In the Loop (HIL) [4]–[6], [10]. SIL focuses on procedures and functions for isolated units and checks for the expected input/output relationships. MIL focuses on simulation models and identification of expected communication flow and interference. HIL focuses on the behavior of the software executing on the target hardware, the use of target hardware-specific functionality, and communication with other subsystems. It is strongly encouraged to use any means possible to enable testing, preferably on target hardware with a focus on early integration [5].

Through this paper, we focus on the testing activities used by the HYPSO-1 team to ensure the desired functionality of the software deployed on the OPU and finally integrated into the CubeSat. A summary of the testing strategies and their focus is given in Tab. I.

III. Embedded Software Testing

Ideally, the payload subsystem is tested in an environment as close to flight as possible. Here, parts of the operation are simulated, e.g., procedures for operation and flight, and only parts of the satellite directly important for the payload are used. Thus, it is not strictly a HIL-system, as we do not have any simulated input from all sensors and outputs to virtual actuators, but rather a Payload-HIL (P-HIL).

Embedded systems are typically tested in different ways throughout various development stages. For the HYPSO-1 mission we tested the functionality and performance of communication and image processing independently. The communication is tested by interfacing the payload with the other physical satellite bus subsystems. For the onboard image processing, the development algorithms usually start as a proof-of-concept prototype in Matlab or Python. If the concept is promising, it is then transformed into a C program and tested on a desktop computer. This code

TABLE I: Testing Strategies, with details in Sec. III-B

Focus	Category
Automated workflows focusing on Continuous Integration (CI) for development and deployment	HIL SIL
High availability of target hardware for development and subsequent testing.	HIL
The development of test suites simulating nominal operations	HIL MIL
Rehearsal with trained and untrained operators	HIL MIL SIL
Post-launch Testing and Development	HIL MIL SIL

is then ported and adapted to the target hardware; where the processor, power, and memory resources are limited and thus constrain the operation and performance. In this stage, target architecture-specific instructions have been used to improve performance when possible. If appropriate, the algorithm may be implemented in a hardware-accelerated form; employing a Field Programmable Gate Array (FPGA). This process is illustrated in Fig. 3.

As shown in Fig. 1, system and integration tests started when the first HIL-setup with the OPU was in place in February 2020. The testing continues as discovered bug fixes and new features are intended to be updated during flight [1].

A. Testing infrastructure

Fig. 4 shows the basic system architecture for the fully deployed system. The Payload Controller (PC) is responsible for propagating commands and responses to and from the payload. The lightweight and broker-less messaging library used in parts of the fully deployed system is nanomsg Next Generation (NNG). The other abbreviations are explained below.

Fig. 5, 6, and 7 illustrates how multiple subsystems of the procured CubeSat platform are accessed. Each colored box has a unique CubeSat Space Protocol (CSP) address for the given test setup. Different communication pathways can be tested by using different CSP addresses through a Controller Area Network (CAN) and internet-based connection to a FlatSat at the satellite platform vendor's premises. The FlatSat is a mimic of the actual CubeSat where components are installed and connected so that it can be interfaced with remotely for testing [4].

The software propagating commands or requests from an operator to the payload uses a service-oriented architecture based on a request-response pattern. That is, the operator sends a request via the Command Line Interface application `hypso-cli` application on the ground and the `opu-services` application running on the OPU sends a response. The satellite bus and its ground system rely upon the use of CSP, and the OPU also implements CSP as its main communication protocol.

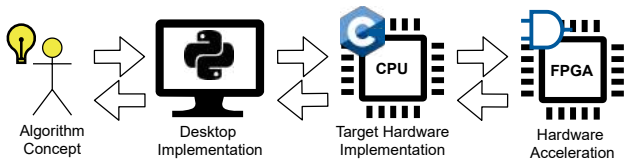

Fig. 3: Illustration of the algorithm development flow. If one stage passes it may move on to the next, or be revisited.

1) LidSat for development: The LidSat-setup is the most complete and versatile setup. It includes the OPU and integrates with the development models and prototypes of the rest of the satellite bus. It also enables testing of communication links by UHF-radio in the test-loop, as well as the Mission Control System (MCS) environment to better emulate the expected communication interfaces between the operator and CubeSat, as seen in Fig. 5. This is also used for the system functional test campaign here called rehearsals (also known as Test-as-you-fly [4], [5]), covered in Sec. III-B.4.

2) P-HIL for automated tests: The Payload-HIL in Fig. 6, featuring only the OPU and an Electrical Power System (EPS), is mainly used for automated testing. The P-HIL setup connects to a testing automation server using Jenkins, capable of running regression tests on Pull Requests (PRs) and after branches were merged, and periodic performance tests, as defined by the requirements. It is also possible to remotely run the Jenkins tests on the LidSat, enabling automatic tests of system functionality [4].

3) Aid for environmental tests: Environmental test setups, i.e. the EM and QM of the payload [4] were used when the payload was tested in the expected thermal and vacuum conditions. A setup like this is given in Fig. 7. To support the environmental test campaigns, semi-automated test scripts allowing controlled repetitions of tests were developed, based on the same tests used on the P-HIL test setup. In addition, an electrical Ground Support Equipment set, consisting of Power Supply Units (PSUs), cable harness for power, and communication was developed, as it was not deemed expedient to test the flight-proven EPS. The PSU in Fig. 7 can be regarded as a simplified and manual EPS.

B. Testing Strategies

Several strategies for testing were attempted, to better understand what worked best for our team. Here are four

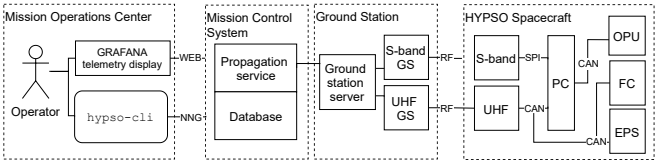

Fig. 4: Fully deployed system architecture. Onboard Processing Unit (OPU), Flight Computer (FC), Electrical Power System (EPS), Payload Controller (PC), Controller Area Network (CAN), Ground Station (GS), Radio Frequency (RF), nanomsg Next Generation (NNG)

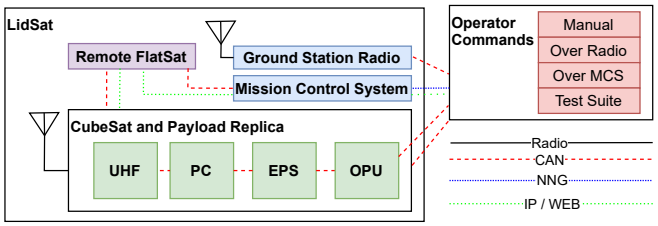

Fig. 5: The LidSat testing setups where different paths for command propagation is used by utilizing unique CSP addresses to test communication at different levels.

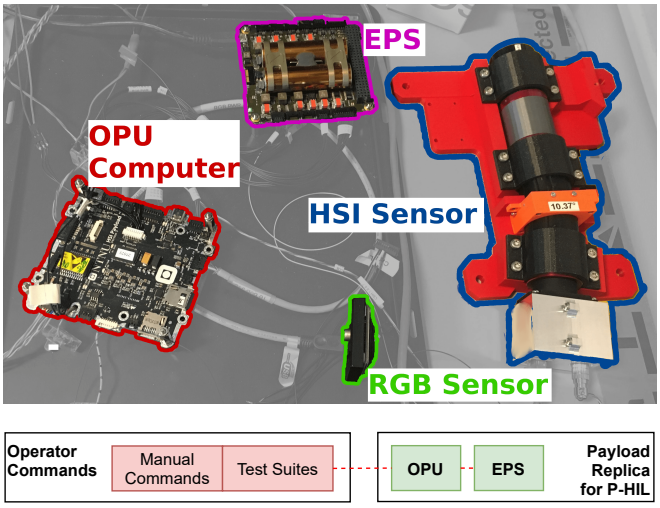

Fig. 6: The P-HIL testing setups where the payload and EPS is accessed via CSP over CAN marked with dashed red line.

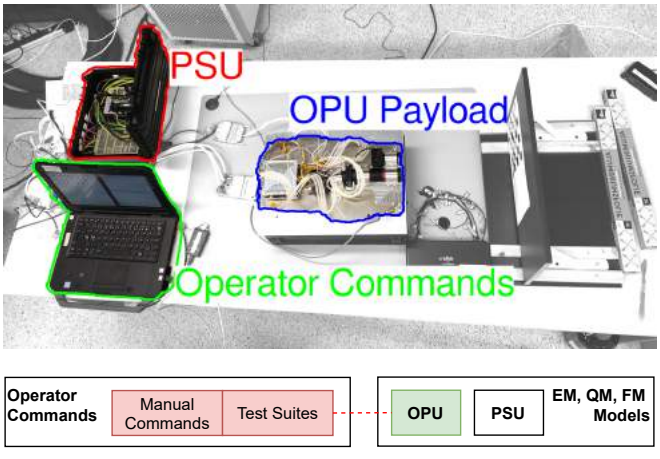

Fig. 7: The testing setups used for environmental validation of the payload where it is accessed via CSP over CAN, marked with dashed red line. The PSU is manually operated during environmental testing.

high-level descriptions of the testing strategies that were eventually chosen.

1) Continuous Integration Workflows: As part of the workflow, procedures and functions are tested using the *Check Unit Testing Framework for C* [11]. That is, at each PR a set of unit tests are executed to check if the input/output relationship for the functions is still as expected. Ideally, the tests would be developed first, followed by a function that meets the defined requirements. Using this development strategy for our team has proven difficult, possibly due to the limited timeframe student developers have, and inexperience in the methodology. As a result, relatively few unit tests have been developed. Regardless, the tests that exist have helped to discover undesirable artifacts.

2) Target Hardware for Development and Test: Ensuring the availability of target hardware for both development and testing has been a priority. Initial testing is done by the developer on the target hardware, which helps to identify basic errors. When the developer regards the code contribution as ready to be merged into the main repository they open a PR. A review of the written code, as well as a test of the resulting executable(s), is then performed by other team members. This will prompt a discussion as to whether or not the code contribution performs as intended and is maintainable. Changes can be requested before approval. The process, further described in [9], has helped us to identify errors and bugs hidden from the initial developer, as well as to clarify misunderstandings in functional requirements. An added benefit is that an overview and understanding of the codebase is distributed among more team members.

This setup made early integration tests possible, which is recommended [5]. These tests done during development, for both new functionality and regression tests, are usually manually executed with the payload connected to other parts of the satellite; either locally or through a remote FlatSat. Infrastructure functionality, e.g., subsystems communication, execution of remote commands, etc., are tested more frequently as a result. These tests focus on high-level functionality and have uncovered Interface Control Document (ICD) issues, functional problems with User Intrerface (UI), illogical programming, and shortcomings of documentation, among others.

3) Test Suites Simulating Nominal Operations: Automated tests, where nominal operations are simulated on the P-HIL, were also made. A *Jenkins*-server builds a new version of the software for every merged PR and tests its performance on target hardware. These test suites can also be invoked on demand. This has helped us uncover changes in performance and changes in ICDs, as well as making consistent testing of nominal operations more available across different communication paths. The P-HIL is configured to be tested with both the LidSat and a remote FlatSat provided by the CubeSat vendor. The set of nominal tests were also used to test the payload during mechanical, thermal, and vacuum testing, ensuring consistent testing procedures.

4) Rehearsals with Operators: In the rehearsals, an operator simulates how the satellite will be interfaced with

during the mission. This includes enforcing constraints such as communication windows, injecting errors on communication links, and letting the operator carry out complicated procedures. Participants consisted of both people familiar with the UI and people who had to rely on documentation. These rehearsals have revealed functional issues and high-level problems, as well as uncovering procedural inaccuracies and illogical UI. It has proven multiple times to be a useful way to test if the satellite and its interfaces behave as desired, if the predicted constraints will hinder operations, and if the documentation is sufficient.

5) Post-launch Testing and Development: The HYPSO-1 satellite is intended to be updated during its lifetime. There are benefits of having a replica of the system on the ground, e.g., to test telecommands and updates in a non-operational environment [5]. We plan to use our existing test setups and strategies to test new software modules and telecommands, train operators and verify updates. It is planned to further extend the existing testing infrastructure, with more replicas of the CubeSat and Payload.

IV. Testing Results

In our team, it is the creator of a unit of software who is responsible for creating unit tests. This can lead to rapid deployment of tests. However, a risk of this approach is to create tests that pass, rather than tests that test the current software for unexpected results.

A majority of the software testing can be automated with automated test setups. The automated testing has helped uncover changes in the Command-Line Interface (CLI), as well as unintended changes in the request-response pattern between `hypso-cli` and `opu-services`. Three different test suites are run and cover nominal operations of the satellite at every code change. These test suites will then give an error if there is an interface change in the commands used for nominal operations. However, there are still some human interactions that cannot be automated. This does not reduce the importance of an automated test setup as it supports rapid deployment and regression testing. The automated test setups, i.e. LidSat and P-HIL, do not necessarily test for new and unexpected states of the system. This can lead to false confidence in the system, as a system can still fail in operations despite passing all automated tests. This potential false confidence supports the use of manual testing, as done during the code reviews. The manual testing should be extended to incorporate new tests of new functionality but this is not done currently.

The issues discovered during code reviews were mainly concerned with whether or not the proposed changes were performed as desired. When doing these reviews functionality unrelated to the original PR was also tested, and some issues could be reported as a result. Problems with illogical UI and limited documentation were also addressed as part of the review. The code reviews became a good platform to discuss the proposed changes and often led to incremental improvements from the original PR. Initially, these code reviews were highly unorganized, but a more coherent and effective process emerged when establishing a bi-weekly three-hour session for code reviews employing *mob programming* [12].

The rehearsals performed in the fall of 2020 uncovered several new issues and areas of improvement for the software. These issues were both quality-of-life changes for the developers and operators, but some were crucial to the mission. The rehearsals invoke more of the full mission infrastructure and use more of the available communication paths than our other tests. This has been helpful to discover changes in communication performance, issues regarding scheduling, planning, and timing, and learn how the operator needs to handle errors during a satellite pass. In addition, since the communication windows have been constrained, the rehearsals have also let operators experience the stress related to performing a set of tasks within each time window. This experience also leads to the re-design of procedures and the creation of more high-level functions that aid the operator to perform tasks more efficiently. The issues resulted in, among others, the following changes:

- Improved redundancy in the power control of the payload.
- More appropriate default timeouts for file transfer between subsystems and ground.
- Improved state acknowledgment recovery during transmission loss.

V. Discussion

One challenge for the existing setup is the limited test coverage. Not all of the software units have tests implemented for them, both with or without target hardware. At the time of writing, we have 8 test suites testing different services with a total of 46 (successful) unit tests. It would be beneficial to be able to test the entire satellite as part of HIL, and our team attempted to get close to this with the use of a remote FlatSat, a local FlatSat (i.e. the LidSat), and partial reconstructions of the expected CubeSat system.

The use of automated testing, i.e. CI, test suites, and the Jenkins server that connects to P-HIL, lowers the barriers to perform testing and ensures repeatability in testing procedures. Within this test configuration, we can perform both regression and acceptance testing for the satellite payload. Setting up this test configuration comes at an initial cost, but makes it easier to perform consistent tests repeatedly. Thus the overall cost was justified. Setting it up properly took 2 months. Adding more testing functionality became easier with this setup and the protocols for creating tests were established.

Manual testing is labor-intensive but has some clear advantages. The tester can determine if the code performs as intended and we have experienced that a different perspective from a third party has proven fruitful. The process has also been used to request important changes, both for UI and for operations, and larger parts of the team became more familiar with the codebase by doing manual testing. It is difficult to quantify the amount of manual testing versus automatic testing that is performed, but more effort is

definitely put into manual testing. During manual testing, we have also started to do these sessions using a *Remote Mob Programming*-strategy, where one *driver* is responsible for sharing their screen and conducting code changes, whilst the other team members provide helpful comments, questions, and perspectives [12].

The most labor-intensive testing performed was the rehearsals, but they also provided very useful insight in terms of operations. Over the development period, we conducted two such rehearsals. During rehearsals, where communication outages were simulated, it became clear that some of the needed functionality for operations had not yet been properly specified. This testing helped us specify and develop that functionality, while at the same time uncovering completely new problems that would not be caught with automated testing or code reviews.

The rigorous and extensive testing strategies used by large space organizations, such as The European Space Agency (ESA) are not always feasible for a small CubeSat team, and they do not guarantee mission success [13]. The testing strategies given here are not as rigorous or as extensive as they could be but can prove adequate for the size of the team and the accepted risk. The rehearsals and the different HIL-like setups lead to a strong foundation upon which to build further tests.

VI. Conclusion

In this paper we have described the testing strategies adopted by the HYPSO-1 team in the development of their first satellite, presented the issues that these strategies uncovered, and discussed the benefits and limitations of these testing strategies.

Certain aspects of the mission have not been tested end-to-end on target hardware before launch. At launch, the planned image processing pipelines are still under development and need more testing. These processing pipelines will extend the capabilities of the payload and are intended to utilize and provide relevant information from and to other assets [1], [2].

The time referencing of the actual system has not been properly tested as the available clock hardware has not reflected the actual target hardware for this part of the system. This is a consequence of relying on a remote FlatSat.

Early testing is important, as it permits more time to resolve issues. Having a testing infrastructure that is available for as many of the developers and reviewers as possible has proven very useful. The automated testing setups and the nominal test suites provide good test repeatability and can be extended with little additional overhead. The rehearsals and manual testing during reviews have helped discover other problems that were not caught by automated testing. Having an operator in the loop when designing software is deemed crucial for mission success, in addition to automated testing.

References

[1] M. E. Grøtte, R. Birkeland, E. Honoré-Livermore, S. Bakken, J. L. Garrett, E. F. Prentice, F. Sigernes, M. Orlandić, J. T. Gravdahl, and T. A. Johansen, "Ocean color hyperspectral remote sensing with high resolution and low latency–the hypso-1 cubesat mission," *IEEE Transactions on Geoscience and Remote Sensing*, pp. 1–19, 2021. DOI: 10.1109/TGRS.2021.3080175.

[2] J. L. Garrett, S. Bakken, E. F. Prentice, D. Langer, F. S. Leira, E. Honoré-Livermore, R. Birkeland, M. E. Grøtte, T. A. Johansen, and M. Orlandić, "Hyperspectral image processing pipelines on multiple platforms for coordinated oceanographic observation," in *2021 11th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2021, pp. 1–5. DOI: 10.1109/WHISPERS52202.2021.9483993.

[3] E. F. Prentice, M. E. Grøtte, F. Sigernes, and T. A. Johansen, "Design of a hyperspectral imager using COTS optics for small satellite applications," in *International Conference on Space Optics — ICSO 2020*, B. Cugny, Z. Sodnik, and N. Karafolas, Eds., International Society for Optics and Photonics, vol. 11852, SPIE, 2021, pp. 2172–2189. [Online]. Available: 10.1117/12.2599937.

[4] C. del Castillo-Sancho, G. Grassi, A. Kinnaird, A. Mills, and D. Palma, "Lessons learned from aiv in esa's fly your satellite! educational cubesat programme," *Proceedings of the AIAA/USU Conference on Small Satellites*, 2021. DOI: https://digitalcommons.usu.edu/smallsat/2021/all2021/232/.

[5] L. Berthoud, M. Swartwout, J. Cutler, D. Klumpar, J. A. Larsen, and J. D. Nielsen, "University cubesat project management for success," *Proceedings of the AIAA/USU Conference on Small Satellites*, 2019. DOI: https://digitalcommons.usu.edu/smallsat/2019/all2019/63/.

[6] C. L. G. Batista, T. Basso, F. Mattiello-Francisco, and R. Moraes, "Impacts of the space technology evolution in the v v of embedded software-intensive systems," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2020, pp. 1749–1755. DOI: 10.1109/CSCI51800.2020.00324.

[7] A. Poghosyan and A. Golkar, "Cubesat evolution: Analyzing cubesat capabilities for conducting science missions," *Progress in Aerospace Sciences*, vol. 88, pp. 59–83, 2017, ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2016.11.002.

[8] M. Jones, E. Gomez, A. Mantineo, and U. Mortensen, "Introducing ECSS software-engineering standards within ESA," *ESA bulletin*, pp. 132–139, 2002. [Online]. Available: https://www.esa.int/

esapub/bulletin/bullet111/chapter21_bul111.pdf.

[9] E. Honoré-Livermore, R. Birkeland, S. Bakken, J. L. Garrett, and C. Haskins, "Digital engineering development in an academic cubesat project," *Journal of Aerospace Information Systems*, pp. 1–12, 2021. DOI: 10.2514/1.I010981.

[10] J. Kiesbye, D. Messmann, M. Preisinger, G. Reina, D. Nagy, F. Schummer, M. Mostad, T. Kale, and M. Langer, "Hardware-In-The-Loop and Software-In-The-Loop Testing of the MOVE-II CubeSat," *Aerospace*, vol. 6, no. 12, 2019, ISSN: 2226-4310. DOI: 10.3390/aerospace6120130.

[11] *Check — unit testing framework for c*, https://libcheck.github.io/check/, (Accessed on 07/28/2021).

[12] D. Ståhl and T. Mårtensson, "Mob programming: From avant-garde experimentation to established practice," *Journal of Systems and Software*, vol. 180, p. 111 017, 2021, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2021.111017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122100114X.

[13] A. Albee, S. Battel, R. Brace, G. Burdick, J. Casani, J. Lavell, C. Leising, D. MacPherson, P. Burr, and D. Dipprey, *Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions*, NASA STI/Recon Technical Report N, Mar. 2000. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2000STIN...0061967A.

ACKNOWLEDGMENT

The authors would like to thank all the MSc. students contributing to testing, especially Esten S. Dalseg. The authors also want to thank Amund Gjersvik for his contribution by providing hardware, infrastructure and solving engineering problems related to making the hardware available for testing and E.F. Prentice for collaboration on the environmental tests. Finally, we thank the numerous reviewers and are grateful for the participation of the CubeSat vendor NanoAvionics and their continuous feedback throughout the project.