https://doi.org/10.1590/jatm.v17.1379 ORIGINAL PAPER

Systematic Testing Approach for Communicating Software Embedded in Nanosatellites Focusing on Interoperability Faults

Carlos Augusto Paiva Lameirinhas Conceição[1,*], Maria de Fátima Mattiello-Francisco[2]

1. Departamento de Ciência e Tecnologia Aeroespacial – Instituto de Aeronáutica e Espaço – Divisão de Mecânica – São José dos Campos/SP – Brazil.
2. Instituto Nacional de Pesquisas Espaciais – Coordenação de Rastreio, Controle e Recepção de Satélites – São José dos Campos/SP – Brazil.

***Correspondence author:** conceicaocarlos1@gmail.com

ABSTRACT

CubeSats, small standardized commercial satellites, have emerged as platforms for carrying scientific instruments and qualifying innovative technologies in space. However, the high mission failure rate during the launch and early orbit phase (LEOP) has drawn attention to insufficient testing in their development process. Unlike traditional satellites, the shortened project life cycle of CubeSat missions often limits verification and validation (V&V) activities. Traditional V&V approaches address interoperability issues late in development, leading to time-consuming rework when nonconformities are identified during system integration. This paper proposes a Scalable Architecture Test System (SATS) to support the verification of interoperability requirements of CubeSats' embedded software early in development. Interoperability models for two communicating software components are specified, from which test cases are automatically derived and software codes are generated to be embedded in programmable boards connected to a CubeSat communication channel. This architecture supports test execution in a model-in-the-loop (MIL) concept to verify requirements and later validate the implementation, when real hardware replaces simulated models. The approach's effectiveness in early detection of faults is demonstrated in the NanosatC-BR2 project developed at the National Institute of Space Research (INPE), allowing the correction of the specification of software components earlier in satellite development.

Keywords: Dependability; CubeSat; Interoperability; Verification and validation; Testing software-intensive subsystem requirements.

INTRODUCTION

Since the beginning of the second millennium, space has become more accessible, with the CubeSat satellite standard rapidly evolving from a capability-building purpose to a standardized commercial platform. These platforms are now mature enough to carry scientific instruments into space and qualify innovative technologies in orbit (Yost and Weston 2024). At present, complex missions are taking advantage of the reduced project development cycle and low cost offered by CubeSat-based platforms. However, despite the increasing number of CubeSat missions (excluding constellations) launched since 2003, comprising 75% of the secondary payloads (PLs) flown in 2014 (Swartwout 2015), the minimum success rate in orbit decreased over time. According to Swartwout's (2019) mission status assessment, presented at the CubeSat Developers' Workshop, missions are categorized into five stages: dead-on-arrival (DOA), early loss, partial mission, full mission, and unknown. The percentage of missions reaching the full mission stage

Received: Jul. 31, 2024 | **Accepted:** Mar. 24, 2025
Peer Review History: Single Blind Peer Review.
Section editor: Alison Moraes

has significantly declined over the years. From 2005 to 2009, 34.9% of CubeSat missions (33 launched) were fully successful. This rate dropped to 25.3% for 158 missions launched from 2010 to 2014. The decline continued further, reaching just 16.1% for the 292 missions launched from 2015 to 2018.The high infant mortality rate of CubeSat-based missions in orbit has been a subject of concern regarding the development process and the experience of mission developers (Swartwout 2016). Agile approaches, typically adopted in the CubeSat development cycle, contribute to the lack of documentation. The absence of reference publications and guidelines forces each new project to develop its own processes, often leading to the repetition of mistakes made by previous teams (Hestad *et al.* 2023). One possible cause of CubeSat failures is an immature verification and validation (V&V) process.

In space projects, the V&V process plays a key role in ensuring mission quality. Space agencies typically adopt standardized V&V processes, such as those recommended by the European Cooperation for Space Standardization (2018) (ECSS-E-ST-10-02c Rev.1 – Verification). However, the traditional V&V process is frequently costly and time-consuming because it usually involves a quality assurance team dedicated to carrying out the V&V plan, which is incompatible with the philosophy of CubeSat-based satellites that envisage a short development cycle and reduced budget. Moreover, difficulties arise when international standards, procedures, and practices from traditional satellite development projects must be adopted by student teams in their local CubeSat projects (Hestad *et al.* 2023). Verification and validation are procedures used to ensure that a product, service, or system meets requirements and specifications, and fulfills its intended purpose. Verification activities are carried out through different strategies such as modeling, simulations, alternative calculations, comparison with other proven designs, experiments, tests, and specialist technical reviews. Validation ensures that the resulting product is capable of meeting the requirements for the specified application or intended use. Design validation is similar to verification, except this time the product is checked under conditions of actual use.

Among the existing V&V techniques, testing is unique in its ability to collect evidence of faults during software execution, whether in a real or simulated environment. The academic community has developed numerous methods, tools, and testing systems to support the specification, implementation, and execution of effective and concise sets of test cases (model-based testing [MBT]) (Binder 2002; Myers *et al.* 2011). Broy *et al.* (2005) is one such approach that systematizes and automates test case generation and execution, facilitating the evaluation of implementation quality and the early detection and treatment of faults in the development cycle (Mattiello-Francisco *et al.* 2012). Model-based testing has proven effective in identifying system failures and reducing testing efforts (Petrenko and Schlingloff 2013). The Scalable Architecture Test System (SATS) environment, proposed in this paper, allows the generation of integration test suites and systematization of their execution in the CubeSat V&V process. The reliability assessment presented in Langer and Bouwmeester (2016), considering six subsystems of a CubeSat, shows that the on-board computer (OBC) significantly contributes to CubeSat mission failures occurring just after launch. This is known as DOA, where the satellite is ejected from its deployed position but never achieves a detectable functional state. According to their assessment, the OBC, electrical power subsystem (EPS), and communication subsystem (COM), including antennas, are responsible for most mission failures occurring from 30 to 90 days after the satellite ejection in orbit. The EPS accounts for the largest share, causing more than 40% of failures after 30 days, followed by the OBC, responsible for 20%, and the COM, which contributes to 16% of mission failures after 30 days. While the CubeSat standard (Helvajian and Janson 2008) contributes to the reduction of V&V activities due to the standardization of the platform subsystems and interfaces, the same cannot be said for on-board software-intensive subsystems (SiS), like OBC and PL subsystems, which are usually developed specifically for each mission. Therefore, a set of V&V activities, particularly those related to PL integration with the nanosatellite platform becomes necessary.

This paper addresses the challenges of testing interoperability requirements between two communicating subsystems (OBC and a given PL) on board a nanosatellite mission. Interoperability testing is the activity performed by developers during the system integration phase (Binder 2002). It aims at verifying that interactive subsystems communicate with each other as expected. Such tests can reveal non-interoperability evidence, which is often a consequence of incomplete specifications, and/or design faults not identified during the testing of each subsystem in isolation. Given the increasing number of software-implemented functions in nanosatellite subsystems, a systematic testing approach for communicating SiS is essential with the aim of detecting interoperability faults early in the development cycle. Communication failures can have a significant impact on the entire mission.

Model-driven engineering (MDE) is a software engineering approach increasingly adopted to support the V&V process of system requirements. One can design complex software systems in the form of a model from which software code can be automatically generated (Schmidt 2006). The approach allows engineers to think about software requirements at a high level of abstraction, without being concerned about the implementation. Model-driven engineering has been successfully applied in various domains, overcoming the challenge of using different models in different phases of project development through model transformations.

Aiming to anticipate interoperability failures in the development cycle of software-intensive communicating subsystems embedded in nanosatellite missions, the systematic testing approach presented in this paper guides the construction of models that express the expected behavior of two communicating software subsystems embedded in CubeSats under the interoperability abstraction. With the aid of transformation model techniques and MDE tools, the source code of each of the communicating software components is automatically generated. The source codes will be executed in the SATS environment for the verification of the interoperability requirements purposes.

The test cases to be used in the verification process are derived from the simulation of the behavioral model of the interoperable subsystems, aided by MBT tools (Mattiello-Francisco *et al.* 2012).

The unexpected behaviors identified during the execution of software codes at SATS by the generated test cases are meticulously documented in a dependability spreadsheet, which was designed based on the dependability taxonomy proposed by Avizienis *et al.* (2004). This taxonomy serves as a powerful tool for assessing faults according to their specific class, type, and group. Through the systematic classification of each fault, it becomes feasible to evaluate potential interrelationships associated with the identified failure and establish effective measures to prevent its occurrence. Furthermore, this classification enables the formulation of systematic mitigations that can effectively address not only the identified failure but also other potential faults related to dependability issues that might arise throughout the system. By implementing this systematic and proactive approach, a comprehensive understanding of the identified failures can be achieved, allowing appropriate steps to be taken to enhance the overall dependability and resilience of the system.

The effectiveness of SATS in the developed approach is demonstrated through a case study. This case study focuses on the V&V of interoperability and robustness requirements regarding the interaction between two SiS onboard a nanosatellite, named NanosatC-BR2. The NanosatC-BR2 is a 2U CubeSat mission developed at the Instituto Nacional de Pesquisas Espaciais/Centro de Pesquisa Espacial Regional Sul (INPE/CRS) (Schuch *et al.* 2017) that aims to collect data from the Earth's magnetic field and to test in-flight the resistance of circuits to radiation. This mission also encompasses qualifying other new technologies designed in Brazil for space applications.

This paper is structured as follows: first, the systematic testing method is presented, followed by a discussion of related works and the proposed Systematic Testing Approach. This approach utilizes the SATS framework (testing architecture plus method) to systematize interoperability tests. Next, the application of the proposed approach in the NanosatC-BR2 case study is discussed, along with the results. Finally, the paper concludes with the conclusions and future work.

METHOD

The developed approach aims to investigate interoperability faults by systematizing both the specification and execution of tests. These tests evaluate the behavior of two embedded SiS, which interact via a nanosatellite communication channel. The tests enable the verification of the interoperability requirements of each communicating SiS at the beginning of the mission development cycle through model simulation. Fault treatment, also referred to in this paper as mitigation, involves improving the SiS requirement specification if unexpected behavior emerges during simulation. This process aims to prevent future failures when the real SiSs are validated during the integration phase.

Based on the interaction between the communicating software of the subsystems under test (SUT), the approach makes use of a Timed Input-Output Automata (TIOA) framework (Desmoulin and Viho 2007) to model and simulate the nominal behavior of these subsystems in interoperability. The underlying assumption on the formal testing approach of communicating

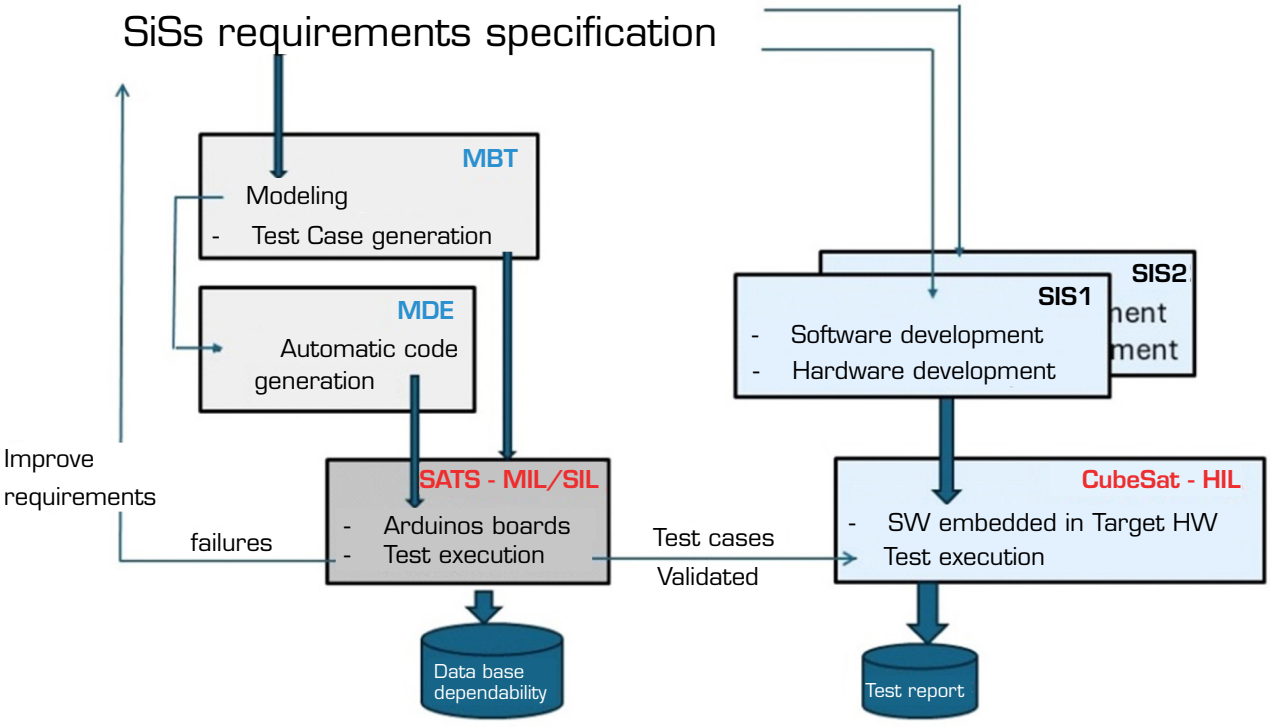
systems is the existence of a precise formal specification model of the SUT and its relationship with the test environment, which is established by SATS. According to Desmoulin and Viho (2007), two principles are the basis for interoperability formalisms: i) interaction verification corresponds to checking that outputs sent by a subsystem (SiS1) on its lower interfaces are foreseen in the specification and that the interacting subsystem (SiS2) is able to receive these messages; and ii) service verification corresponds to the verification that outputs observed on the upper interfaces of the SUT are described in their respective specifications. Thus, outputs must be verified on both upper and lower interfaces, while inputs are controlled only on upper interfaces.

The systematic testing method uses the MBT approach to simulate the behavior of the subsystems modeled in interaction. The execution of the subsystems modeled to interact enables the derivation of test cases, which will be useful for validating the behavior of the communicating SiSs embedded in the CubeSat real boards in the system integration phase.

Another approach used to complement the method is MDE, whose objective is to automatically generate source code for each SiS modeled in interoperability. These source codes are embedded into two different programmable computer boards that compose the SATS configuration for model-in-the-loop (MIL) testing.

The scalable concept of the proposed systematic testing method comprises two branches, as shown in Fig. 1, supporting both MIL and simulated software-in-the-loop (SIL) testing, which uses the SATS environment, and hardware-in-the-loop (HIL) testing, which uses the physical CubeSat environment to validate the real SiSs implementation in interoperability. Model-in-the-loop enables verifying whether the interoperability model of the SUT behaves as expected, at a high abstraction level. Software-in-the-loop enables the execution of the SiS codes embedded in simulated hardware, validating, in the communication channel, the test cases generated automatically. Hardware-in-the-loop validates the final behavior of the real subsystem (hardware/software) in the communication channel, reusing the validated test cases (Srinivas *et al.* 2014).

The method uses the dependability taxonomy of Avizienis *et al.* (2004) to classify possible faults identified during the testing of the interoperability requirements between the communicating subsystems since the requirements elicitation. The information related to both the identified faults and the mechanisms used to mitigate them is saved in the SATS database, being included in test cases.



Source: Elaborated by the authors.

Figure 1. MIL/SIL and HIL systematic testing method.

In addition, the scalable architecture has the potential to support fault injection, emulating failures in the communication channel, for robustness requirement verification purposes (Batista *et al.* 2018). This allows anticipating evidence of unexpected behavior of a particular SiS in the communication early in the SATS environment, and consequently improving the specification of the subsystems to add robustness requirements, for instance.

RELATED WORKS

The proposed approach (the method described above plus SATS) relies on the context of CubeSats' V&V process, more precisely on the challenges of testing interoperability of two communicating SiS embedded in nanosatellites. To the best of the authors' knowledge, few reports on the V&V process addressing CubeSats' software requirements and testing are available in the literature (Silva 2024). Contributions on the use of modeling approaches and tools to support the process are found in certain articles. In the following, a short account of selected references is given.

- Abrahão *et al.* (2017): this article presents the use of the MDE approach and tools for project development, discussing the advantages and disadvantages of using these resources for developers. The article reports that the use of standards and development methods according to the user and the project can be an approach that MDE should also cover. The authors present the use of the Papyrus and DOORS applications to generate models and code. Simulation tests are carried out, but the equipment that will be onboard the CubeSat is not tested.
- Batista *et al.* (2018): the article presents a test architecture called fault emulator mechanism (FEM), useful for robustness testing of communicating SiS onboard nanosatellites. The test architecture supports fault injection in the communication channel. It does not specify what mitigations should be made when a failure occurs or if other failures may occur due to their interrelation.
- Duarte *et al.* (2020): simulates an autonomous redundant attitude determination system for CubeSats using the QUEST modeling method and components (commercial off-the-shelf [COTS]). The system redundancy is tested with a bit-flipping fault injection methodology. The tests do not verify if the PL requirements are in accordance with the specifications.
- Souza and Carvalho (2021): tests the drive that will be used in the attitude controller of a CubeSat. For this purpose, it uses a device under test (DUT), which executes the test code. It uses a double device, which performs the hardware that will be embedded in CubeSat. A personal computer (PC) will execute the test code. Interoperability with the onboard computer and other PL specifications are not tested.
- Rizza *et al.* (2022): designed, validated, and tested algorithms for the attitude and orbit control system of a CubeSat using MATLAB and Simulink applications. A camera-in-the-loop is used in the test. However, interoperability between other CubeSat components is not tested.
- Ciacchella *et al.* (2024): based on the model-based systems engineering (MBSE) methodology, it uses the Capella and TASTE tools to model the finite state machine (FSM) of a CubeSat. It also verifies the software requirements and validates its structure. No robustness tests or validation of the equipment to be used in the CubeSat are presented.

One can observe that these research efforts address in isolation software requirement modeling, automatic generation of software code, fault injection for non-functional software requirement testing, but do not cover the entire V&V process. It is observed that the assessed articles do not address MIL, SIL, and HIL together to support the requirements V&V of CubeSat SiSs. They also do not present ways to catalog the identified failures and the proposed mitigations.

The FlatSAT concept (Amason 2008; Barcellos *et al.* 2023) inspired the SATS to systematize the V&V process of SiS. FlatSAT has usually been adopted in CubeSat-based missions to support the preliminary integration of PL with the other subsystems of the satellite. It also helps validate system requirements in the satellite engineering model at the system integration level, which occurs late in the project development cycle. However, less attention has been given in FlatSAT to the interoperability requirements implemented by software embedded in the subsystems. Interoperability faults, caused by incomplete specifications and/or design errors, are often not identified during the testing of each subsystem in isolation and will be revealed late in the development cycle, leading to costly rework.

The systematic testing approach presented in this article extends the FlatSAT concept to the embedded-software development cycle, adding modern techniques and tools (MDE and MBT) for systematizing test case specification, validation, and reuse. Moreover, the proposed SATS architecture is based on COTS boards and can be reconfigurable to support MIL, SIL, and HIL software testing phases. This systematic approach allows for the anticipated detection and prevention of faults through software testing materialized early in models' execution, late in code embedded in emulated hardware, and finally in the end code embedded in the real hardware.

Another contribution of SATS to the V&V process is the establishment of a faults and mitigation database that records lessons learned for use in future projects, in addition to being structured according to a well-known dependability taxonomy (Avizienis *et al.* 2004).

SYSTEMATIC TESTING APPROACH

The proposed approach uses the SATS framework (testing architecture plus method) to systematize interoperability tests between SiS communicating in nanosatellite space missions. The approach is structured in six modules and encompasses the validation and reuse of tests in a systematic way. The modules cover the V&V process using the concepts MIL, SIL, and HIL described previously in the Method section.

Figure 2 presents the SATS framework. The notation used in the figure should be interpreted as follows: i) the first number in Arabic numerals identifies the test module to be performed; and ii) the second number in Roman numerals identifies the procedure within the referenced module.

The SATS framework supports the testing process covering the test suite specification from behavioral models execution; the test cases validation in simulated test environments, and the test suite reuse in the integration and system phases of SiSs on board CubeSat for requirements V&V purposes. The six modules follow an evolutionary process that delivers a set of effective tests for the specified subsystem requirements (SiS), allowing the output of one module to be enriched and validated in the subsequent module.

MODULES DESCRIPTION

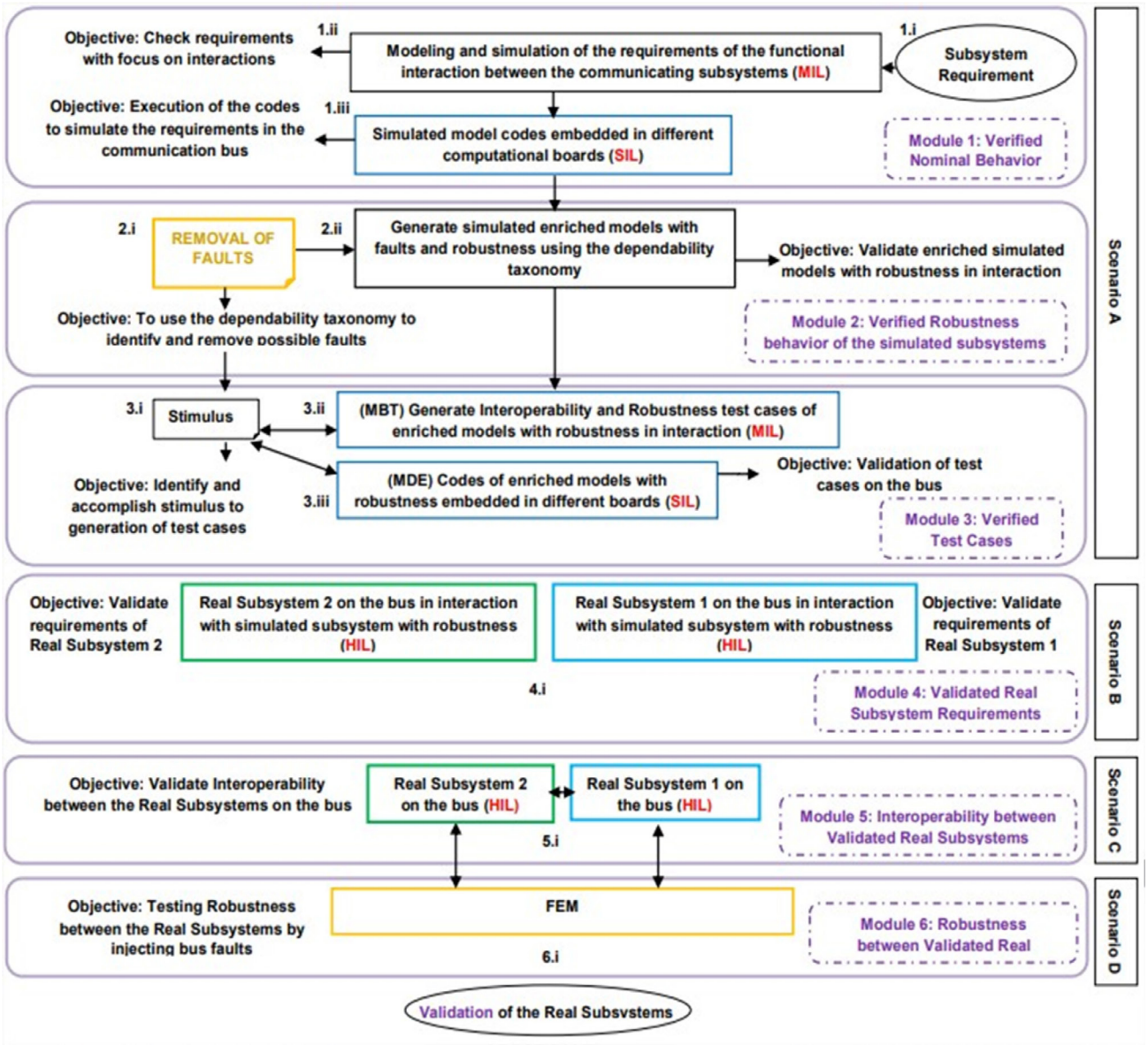
The approach enables reusability in an evolutionary manner throughout the testing process. The evolutions cover the six main test modules, which have been defined to meet the objectives listed below:

- Verify the functional nominal behavior of the communicating subsystems with a focus on their interactions;
- Verify the robust behavior of the communicating SiS, i.e., the capability of a system to operate correctly in unexpected or abnormal known conditions;
- Verify test cases in the robustly validated subsystems;
- Validate the requirements of the real subsystems separately;
- Validate the interoperability of the real subsystems in the bus;
- Validate the robustness of the real subsystems by injecting faults into the bus.

The test modules are described according to Fig. 2 as follows

Module 1 is meant to verify the nominal behavior of the subsystems in a simulated implementation. To achieve this, the following procedures are performed: i) specify the functional requirements for the interaction of the communicating SUT; ii) model and simulate the requirements of the nominal functional behavior using TIOA formalism (MIL); iii) execute the software in a simulated environment (simulated SIL) using the MDE approach concept to generate the source code of the subsystems for a prototype.

Module 2 aims to verify the robustness behavior of the subsystems in a simulated implementation. To achieve this, the following procedures are performed: i) use the dependability taxonomy of Avizienis *et al.* (2004) to identify and classify possible faults that



Source: Elaborated by the authors.

Figure 2. Systematic testing in six modules.

may occur in the interaction between the communicating SUT and report mitigations to avoid them; ii) enrich the simulated models with faults and mitigations, generating models enriched with faults and models that include the robustness behavior – the fault-enriched model of one subsystem is used to test the robustness of the other subsystem and then the roles are reversed.

Module 3 is designed to verify the test cases, including robustness on the communication channel (also referred to as the bus). To achieve this, the following procedures are performed: i) use the identified failures as a stimulus to generate test cases; ii) use the MBT approach to automatically generate test cases, aiming to verify the interoperability and robustness of models including robustness behavior (MIL); iii) use the MDE approach to automatically generate and embed the codes of models including robustness behavior in the different boards connected to the bus (SIL).

Module 4 is intended to validate the requirements in real subsystems separately. To achieve this, the following procedure is performed: connect one of the real subsystems to the bus (HIL) together with the simulated (prototype) version of the other

subsystem that includes robustness behavior; then, the roles are reversed, i.e., the simulated version of the first subsystem is connected together with the production version of the second subsystem.

Module 5 is focused on validating the interoperability between the real subsystems. To achieve this, the following procedure is performed: connect the two real subsystems to the bus.

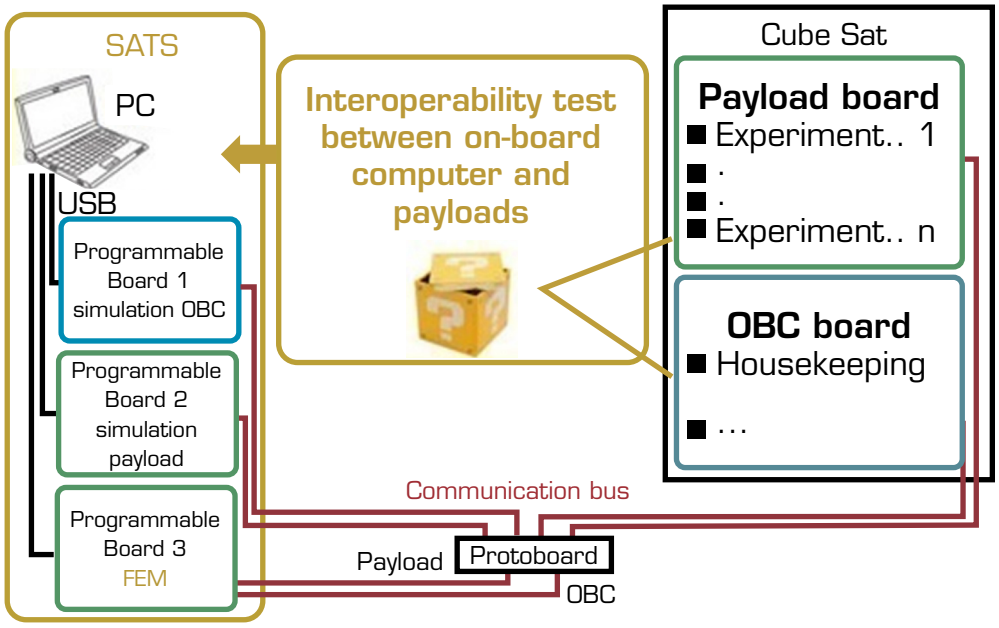
Module 6 is meant to validate the robustness of the real subsystems. To achieve this, the following procedure is performed: inject faults into the real subsystems using a FEM (Batista *et al.* 2018) connected between the subsystems. This mechanism receives information from one of the subsystems, injects a fault, and sends the changed information to the target subsystem. This fault injection in the communication channel allows for simulating the occurrence of failures. Some of the failures include message delay during information exchange, bit exchange, bus delay, and others, which can be registered in a database that contains failure stimuli. The objective is to validate the robustness of the subsystems under fault conditions. In the previous test, it was possible to introduce faults by means of the simulated models. Since in this test, both systems are real (i.e., production) implementations, the faults need to be injected by fault injection, which is performed by the FEM module.

SCALABLE TEST ARCHITECTURE

The architecture of the developed approach uses COTS programmable computer boards. The architecture consists of:

- A programmable board to emulate the OBC;
- A programmable board to emulate PLs;
- A programmable board for injecting faults to verify the behavior of CubeSat subsystems under normal and adverse conditions;
- Boards of the actual subsystems of the satellite under test (OBC and PLs);
- A PC with the necessary development environments to carry out the evolution of the SATS tests;
- A "protoboard" for modeling experimental circuits.

Figure 3 shows the test architecture, which supports four test scenarios as shown in Fig. 2. The scenarios involve tests of MIL, simulated SIL, HIL, and robustness.



Source: Elaborated by the authors.

Figure 3. Test architecture with SATS.

TEST SCENARIOS

Modules 1, 2, and 3 are performed in scenario A, in which two programmable computational boards, boards 1 and 2, are used to simulate the functional nominal behavior of the two subsystems modeled in interaction on the bus. Module 4 is performed in scenario B, in which each real subsystem is tested separately on the bus in interoperability with a board containing the source code of the corresponding simulated subsystem with which it interacts. That is, a test is performed with the real OBC in interoperability with a simulated PL, and the other test involves a real PL with the simulated OBC. Module 5 is performed in scenario C, in which the real subsystems are tested in interoperability on the bus. Module 6 is performed in scenario D, in which the robustness of the real subsystems is tested by injecting faults using a board containing a FEM connected to the bus between the real subsystems. The functional interoperability requirements of the OBC with CubeSat PLs are tested in the communication bus.

FAULT TREATMENT

One of the key activities of the presented approach is the fault treatment activity, which is performed within Module 2 (see Fig. 2). The objective of this activity is to identify and treat possible faults that may occur during the interaction of the SUT. The activity of fault treatment consists of the following procedures:

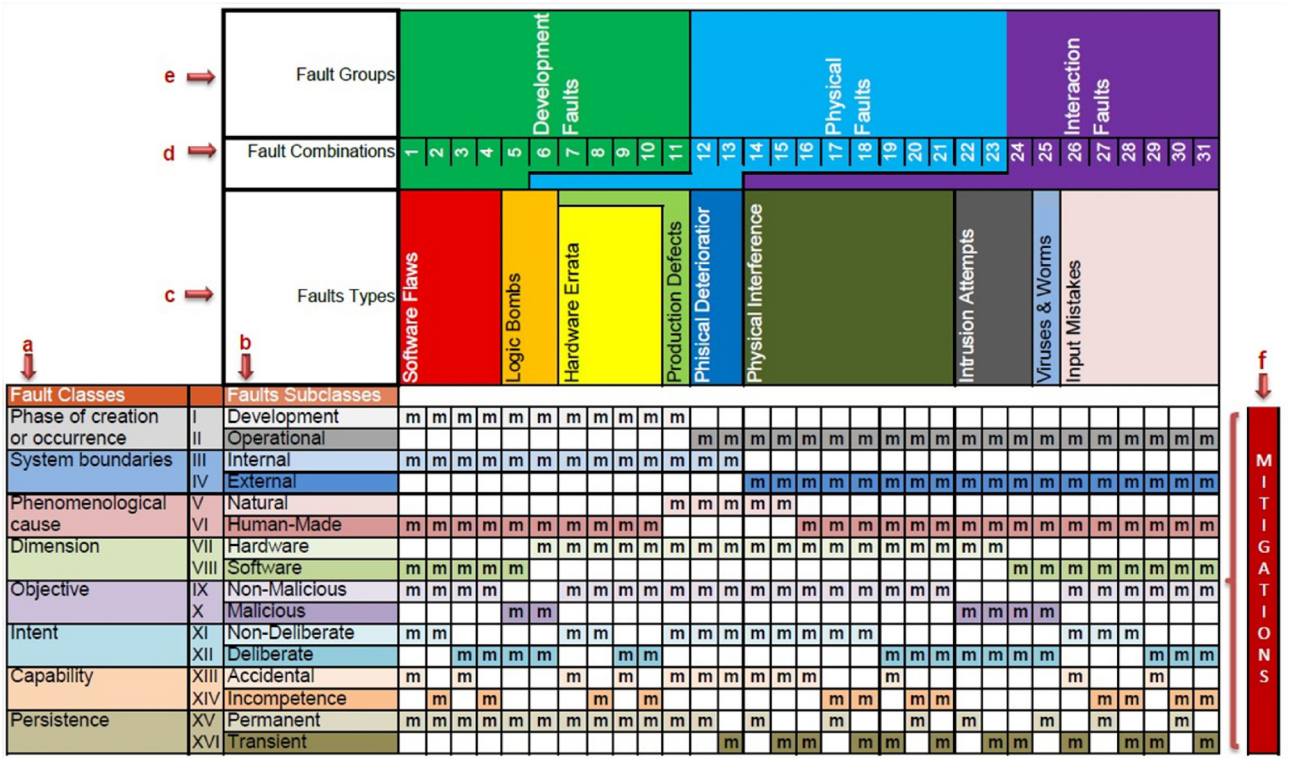
- Dependability worksheet: used to analyze the SiS requirements specification, which is modeled in scenario A, in terms of identifying unspecified operations that may cause undesirable behavior, and are regarded as faults in the specification. Report such faults in the dependability worksheet, which is based on the dependability taxonomy proposed by Avizienis *et al.* (2004), along with their respective classes of faults.
- Faults and effects table: used to record the identified faults and the mechanisms and procedures required to avoid them.

DEPENDABILITY WORKSHEET

The objective of this spreadsheet is to correlate each possible fault identified, classified according to the dependability taxonomy proposed by Avizienis *et al.* (2004), with the possible specification of additional requirements. These additional requirements contribute to mitigating the fault in one or both interoperating SiS. The implementation of such requirements aims to mitigate the fault. Within the approach, the spreadsheet is useful for guiding the enrichment of the nominal interoperability model with the new selected mitigation requirements (MRs), generating the enriched model with the fault MR (FMR) (see Module 2).

Referring to the fault taxonomy matrix of Avizienis *et al.* (2004), the dependability spreadsheet presented in Fig. 4 contains the following information: a) eight fault classes subdivided each into two subclasses; b) 16 fault subclasses; c) nine fault types; d) 31 fault combinations; e) three fault groups; and f) cells named "m" to record the MRs specified to prevent the identified fault. Each identified fault may have one or more MRs. Figure 4 gives the definition of 31 faults (columns), each characterized as a combination of 16 fault subclasses (rows). There are cases where a fault combination does not occur with a fault class. In this case, the cell of the dependability spreadsheet is represented as blank, therefore there will be no mitigation. An example would be a fault in the software fault type in combination (1), and this fault will not be related to the natural fault subclass (V). In this way, the dependability taxonomy presents 256 different combined fault classes applicable for mitigation, which are indicated by the cells of the spreadsheet with the letter "m." In these cells, the MRs to be implemented to prevent the identified faults will be registered.

Extending the Avizienis' matrix for mitigation purposes, the proposed method systematizes the use of the cells on the dependability worksheet. These cells are used to register new MRs that are specified to enrich the interoperability models that exhibit nonconformance behavior during the verification. Thus, the cells of the dependability worksheet can be incrementally populated with new properly named MRs for all verified interoperability models. It is worth highlighting that each cell is not limited to housing just one MR and more than one MR, may be specified to prevent a given fault.



Source: Adapted from Avizienis et al. (2004).

Figure 4. Dependability spreadsheet.

The registration of identified faults and their respective mitigations, in accordance with the dependability spreadsheet, generates a database that allows the traceability of the fault prevention requirements tested during the CubeSat tests. The database represents the memory of faults/mitigations in the history of the use of the SATS environment for verification of requirements in previous tests, serving as a reference for preventing faults in future tests of the same mission or similar missions.

FAULTS AND EFFECTS TABLE

This table will initially be composed of information about possible faults in the worksheet. Based on this information, the consequences of the faults during the tests are reported, together with the mechanisms that can identify and/or avoid these failures. The information that will make up this table is: a) procedure for identifying possible faults; b) fault subclasses, fault combinations, and faults identified; c) effect; d) affected location; and e) control mechanism.

As the tests are performed, new information is added to this table in an iterative process, allowing requirements to be further analyzed, modeled, and tested, thus improving the ability of the approach to identify faults in the real system.

CASE STUDY - APPLICATION OF THE APPROACH IN THE NANOSATC-BR2

The INPE (2015) has been conducting research on nanosatellite platforms, aiming to acquire knowledge, develop new space technologies, and qualify them in orbit. Instituto Nacional de Pesquisas Espaciais, together with its partners, developed a 2U CubeSat called NanosatC-BR2 (Schuch *et al.* 2017). Its mission includes collecting data from the Earth's magnetic field and testing the resistance of circuits to radiation, among other new technologies designed in Brazil.

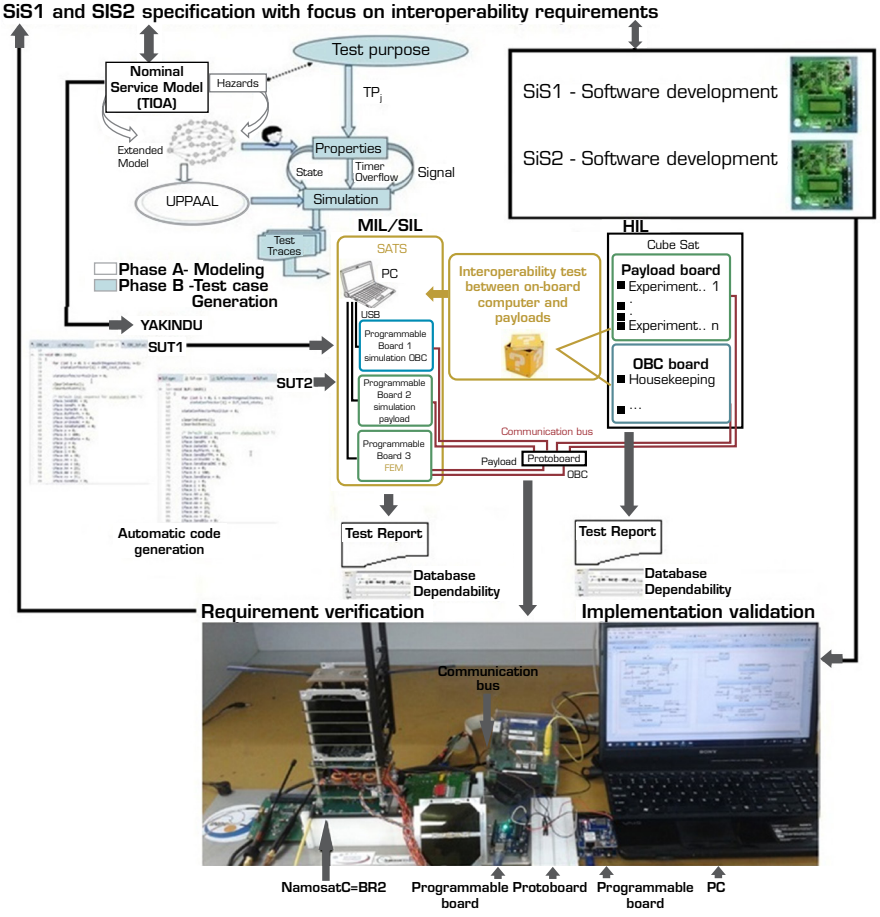
The architecture of NanosatC-BR2 is composed, among other components, of the following subsystems: i) OBC; ii) power source; iii) TRXUV VHF/UHF transceiver; iv) antenna system; and v) solar panel. The PLs are: i) Langmuir probe (SLP) (which measures the numerical density and the spectral distribution of plasma irregularities); ii) Santa Maria Design House (SMDH) a specific application of an integrated circuit (ASIC); iii) field-programmable gate array (FPGA) (an array with configurable ports); iv) magnetometer (which measures the intensity of the Earth's magnetic field); and v) determination attitude system (DAS) (Schuch *et al.* 2022).

Interoperability between the OBC subsystem and satellite PLs is achieved through the I2C communication bus.

The systematic testing approach presented in this paper has been proposed to support the requirements V&V of SiS, including PLs, in the different phases of the development cycle of a nanosatellite. The approach was applied in the development of NanosatC-BR2, specifically for interoperability V&V of the OBC and one of the PLs , called the SLP, referred to as SiS1 and SiS2.

Figure 5 synthesizes the entire systematic testing approach, as presented in Fig. 1, instantiated in the NanosatC-BR2 case study. The test architecture SATS comprises the programmable boards (Arduino) connected to the boards of the satellite through the protoboard to support MIL/SIL. Figure 5 also highlights the artifacts used in the testing process, including the delivered reports that are uploaded into a database based on dependability concepts. The engineering model of the satellite NanosatC-BR2 is part of SATS, as one can observe in Fig. 5.

The application of the systematic testing approach in the NanosatC-BR2 case study followed the six modules introduced in the systematic testing method and presented in Fig. 2. Each module is described below.



Source: Elaborated by the authors.

Figure 5. Test architecture SATS.

RESOURCES USED

In the approach, the following development resources are used:

- The modeling tool selected for the MBT implementation is Uppaal University (2024);
- The MDE modeling environment used is YAKINDU Statechart Tools for Arduino (Itemis 2024), allowing the automatic generation of source code for Arduino computational boards;
- The programmable computer boards used are Arduino because they are cost-effective, and several free libraries are available for these boards, facilitating the implementation of resources such as the communication protocol;
- The libraries for Arduino used are I2C protocol, I2C logic analyzer, Ethernet and database access. These libraries are inside the Arduino IDE.

MODULE 1

In order to apply the proposed approach, some basic requirements for the interaction between the two subsystems were analyzed from Almeida (2016) and are reported as follows:

- The OBC shall send commands to verify the SLP's operating state;
- The data of the experiment stored in a buffer of the SLP shall be read by the OBC and recorded in its memory area to be transmitted to the ground station;
- After transferring the data from the SLP buffer to the OBC memory, the SLP shall perform the acquisition of new data;
- The exchange of information between these two subsystems shall be through the I2C communication bus.

Specification of requirements

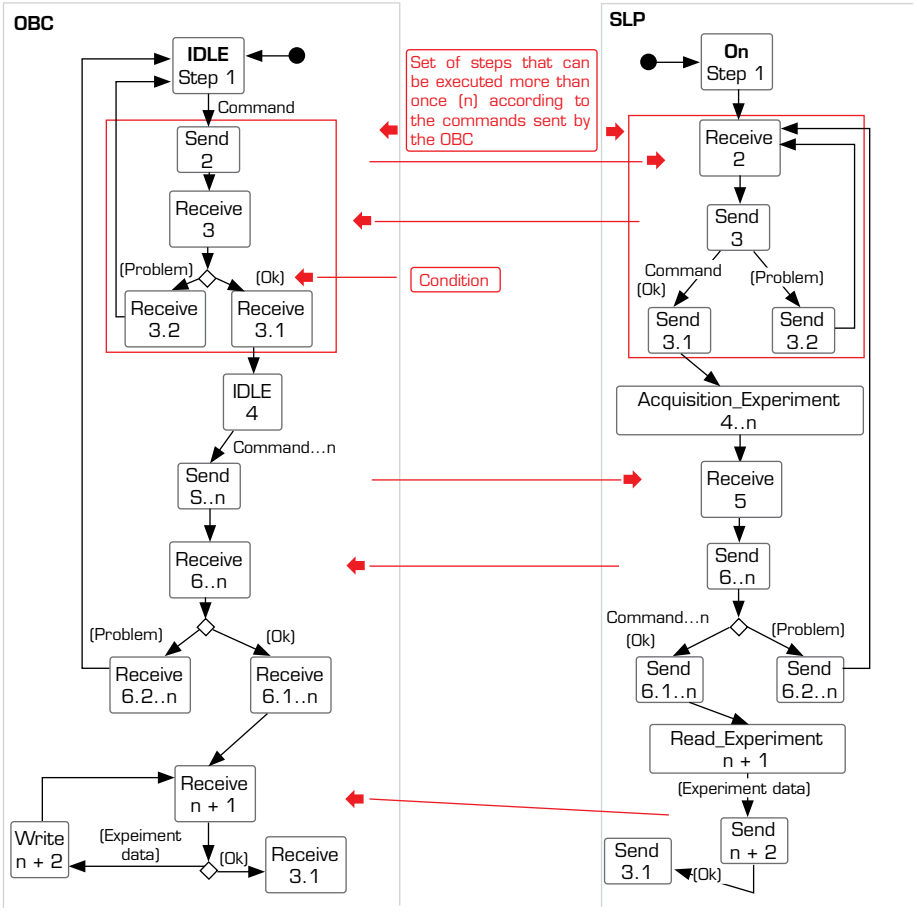
The proposed approach uses, as input, the requirements of the communicating subsystems. The specified basic requirements of these subsystems for conducting the nominal behavior model were:

- The OBC subsystem assumes the role of master, and its nominal behavior in the interaction with the SLP PL shall be represented by the following steps: i) be in operation, waiting for information from the SLP (IDLE step); ii) send a command to the SLP (Send step); iii) receive information from the SLP to define the next operation (Receive step); and iv) write in the memory area the data read from the SLP experiment (Write step);
- The PL subsystem SLP assumes the role of slave, and its nominal behavior in the interaction with the OBC subsystem shall be represented by the following steps: i) be in operation (ON step) while waiting to begin communication with the OBC; ii) receive OBC command (Receive step) to evaluate the next operation to be performed; iii) send information requested by the OBC (Send step); iv) acquire the data of the experiment and store it in the buffer (Acquisition_Experiment step); and v) read the experiment data (Read_Experiment step) and send it to the OBC (Send step).

Other information of interest for the purpose of this paper is the exceptional conditions between subsystems. The following conditions are focused on: i) if the OBC receives a "no acknowledgment" information message, it shall inform that there was a problem and resend the first command; ii) if the SLP receives a "no acknowledgment" command, it shall read the first command returned by the OBC again; iii) if the SLP acknowledges the command received from the OBC, the SLP shall inform the OBC that the command received is OK; iv) when the SLP finishes the acquisition of the experiment, it shall inform that it is ready to read a new command and send the experiment data to the OBC; and v) when the SLP finishes sending the data to the OBC, the SLP shall inform the OBC that it is OK and return to the acquisition of new data for the experiment.

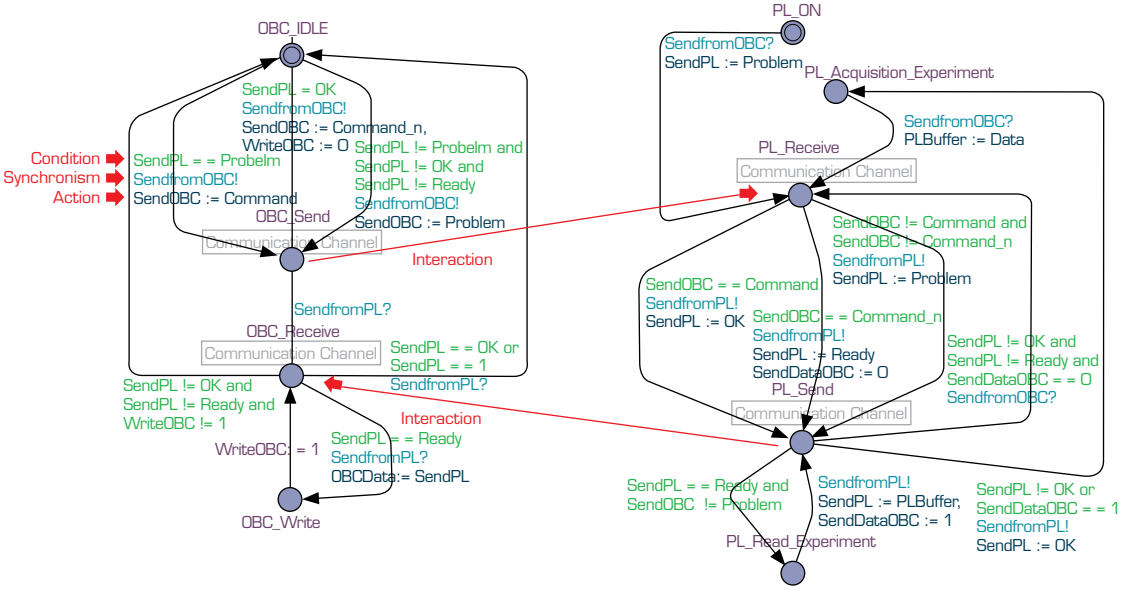
Figure 6 presents the activity diagram of interoperability between the OBC and SLP. The long (red) arrows between the OBC and SLP indicate the synchronization of the interaction between them, that is, at each step, a subsystem sends or receives information from the other, at the moment of interaction.

In the OBC and SLP subsystem diagrams, Send activities take place at the steps in which these subsystems send information. Receive activities take place at the steps in which these subsystems receive information. Focusing on the interaction between the OBC and SLP, this diagram can be restructured by representing only one Send activity and one Receive activity in each diagram. It identifies the input and output situations as modeled and presented in Fig. 7.



Source: Elaborated by the authors.

Figure 6. Activity diagram of interoperability between OBC and SLP.



Source: Elaborated by the authors.

Figure 7. Modeling of OBC and PL subsystems in interaction.

Modeling and simulation

To model the subsystems in Uppaal, some variables were defined for subsystem actions, storage, and data transfer. The defined variables were: i) WriteOBC (write data from the experiment); ii) OBCData (receive data from the experiment); iii) SendDataOBC (send data to OBC); iv) PLBuffer (store experiment data); v) SendOBC (send information from OBC to SLP); and vi) SendPL (send information from PL to OBC).

The Uppaal modeling tool used in the approach performs the synchronization between state machines using the following mechanism: i) a channel identifier followed by the exclamation character (!) is defined as an output action, i.e., writing information in the channel; ii) a channel identifier followed by the interrogation character (?) is defined as an input action, i.e., reading information on the channel. Write and read actions on the same channel generate synchronization. In the modeling, the synchronization was performed through the SendfromOBC! for OBC to send a command to SLP and SendfromPL? to receive information. Figure 7 presents the modeling of the OBC and SLP subsystems in the interaction specified in Uppaal.

In the simulation of the interaction between the OBC and PL, the requirements of the nominal functional interoperability behavior between these subsystems are being realized according to the specifications.

Transformation of models

In the presented approach, the YAKINDU tool is used to generate the automated source code of the OBC and SLP models in interaction, based on the models represented in the Uppaal tool. The objective is to embed the codes generated on computer boards and execute these codes to simulate the functional behavior of the communicating subsystems interacting on the communication bus (SIL).

In this model transformation step, the representations of the states, transitions, actions, and conditions in Uppaal are directly mapped to the corresponding elements in the YAKINDU model. The changes made in YAKINDU modeling are as follows: i) removal of the synchronization defined in Uppaal; ii) using the choice feature in the transition of the state machine to check if the condition is being performed; iii) representing the condition in brackets []; and iv) placing the action on the state transition after the "always" command.

The graphical representations and simulations, both in Uppaal and YAKINDU, have made it possible to verify at this early stage of project development if the information flow and requirements were being satisfied according to specifications.

Some verification at this stage of the evolution of the tests allows identifying possible unspecified faults. An example of a fault is, in the case of one subsystem sending unspecified information; the other subsystem will send a "no acknowledgment" command. However, the subsystem that sent the information may be sending the same information, and this situation may remain for an indefinite period if there is no mitigation for this type of fault.

MODULE 2

The dependability taxonomy of Avizienis *et al.* (2004) was used in the approach for the development of a dependability worksheet and a fault and effects table. This made it possible to classify possible faults in the interaction between the communicating subsystems, identify possible faults related to dependability, and relate mitigations to avoid the identified faults.

Faults treatment

The identified faults were cataloged in the dependability worksheet and in the faults and effects table, as well as the mitigations required to avoid these faults. One fault example presented above can be classified in the dependability worksheet, Fig. 4, under Subclass I (Development), Fault type (Software Flaws), Development Fault Group, and Combination 1.

The simulated nominal model of the SLP was enriched with the cataloged fault example (sending unspecified information to the OBC), creating a model of the SLP enriched with the fault.

The simulated nominal model of the OBC was enriched with a mitigation to avoid the reported fault example. The mitigation performed in the OBC was that, in case it consecutively receives unspecified information three times, it should trigger an error indicator, allowing it to be analyzed by the ground station and take the necessary procedures. The mitigation enriched in the nominal OBC model generated the OBC model that includes robustness behavior.

The other identified faults were also cataloged in the dependability worksheet with their respective mitigations. These faults and mitigations were implemented in models with faults and in the respective models with robustness behavior.

Mitigations were implemented in the nominal model of the SLP, thus producing a model that includes robustness behavior.

The objective was to enrich simulated OBC and SLP models with robustness, and these models were tested through the models enriched with faults.

MODULE 3

In this test evolution, the faults identified in previous tests are used as stimuli to generate test cases using the Uppaal tool. The stimuli are implemented in a subsystem model with the purpose of verifying the interoperability behavior of the other subsystem already enriched with robustness. This test generates test cases to verify if the model has the necessary robustness to avoid the fault stimulus.

In this evolution, the stimulus to generate a test case was the SLP sending an unrecognized message. The robustness implemented in the OBC was to try to recognize the message for three attempts. If the message remained unrecognized after these attempts, it would trigger an alert to be identified by the test team to take the necessary measures.

The stimulus generated the test cases, demonstrating that the implemented robustness successfully identified and avoided the interoperability fault among the tested subsystems.

MODULE 4

For the implementation of this module, the actual OBC subsystem was connected to the communication bus (HIL) along with the board containing the simulated SLP, which was enriched with robustness. The objective was to verify if the real OBC subsystem, in interaction with the validated SLP prototype, met the requirements and did not present interoperability faults on the bus (HIL).

RESULTS

After the evolution of the tests and implementation of the HIL on the communication bus, some procedures for checking and detecting faults during the performed tests are listed below:

- Procedure 1: send an OBC (master) command to the SLP (slave) and check for communication on the bus and whether the information sent by the master has been received by the slave.
  Method 1: send commands through the OBC software developer environment.
  Use an oscilloscope or logic analyzer to check traffic on the I2C bus.
  Verification 1: fault. There was no information traffic on the bus.
  Solution 1: change data transmission speed between subsystems.
  Note: this fault was identified during the test´s evolution in the fault treatment and reported in the dependability worksheet in Subclass XV (transient), fault type (physical interference), and group 15 (interaction faults), as will be presented later.
- Procedure 2: send a sequence of commands from OBC to SLP as specified in the requirements.
  Method 2: send commands through the OBC software developer environment.
  Use an oscilloscope or logic analyzer to check traffic on the I2C bus.
  Verification 2: fault. Some commands did not execute in the expected sequence.
  Solution 2: analyze the flow of information in the OBC subsystem according to requirements specifications.
  Note: this fault was identified during the tests' evolution in the fault treatment and reported in the dependability worksheet in Subclass I (development), fault type (software), and group 1 (development). During the evolution of the simulation tests in the approach, the flow of information according to the requirements' specifications was verified. The simulated model was according to the requirements specifications.

The faults identified in verifications 1 and 2 were reported in the faults and effects table. This table has been filled with information regarding the diagnostics of these faults. Table 1 presents this information.

Table 1. Faults and effects table.

a) Procedure identifying possible fault	b) Fault subclasses/ combination/ identificated fault	c) Effect of fault (consequence)	d) Location affected by fault	e) Control mechanisms for fault detection
1. Sending a command from OBC (master) to the PL (slave)	I/15/.1	Oscilloscope receives inconsistent SDA and SCL signals	OBC and PL did not exchange information	Oscilloscope and voltage regulator
2. Sending a commands sequence from OBC to PL	I/1/.1	Logical analyzer captures SDA and SCL signals according to I2C protocol, but some information is not transmitted in the specified sequence	OBC and PL did not exchange the specified information	Logic analyzer I2C

Source: Elaborated by the authors.

The first test procedure of this table was analyzed to verify if the detected fault and the mitigation to be performed were predicted in the dependability worksheet and was performed during the approach evolution. It was found that the detected fault was related to Subclass XV in the dependability worksheet, and the mitigation to avoid this fault was also listed in this worksheet as shown in Table 2.

At this point, it is analyzed whether the process for mitigating the possible faults was performed. In case some step was missed, it is recommended to re-analyze the requirements.

Table 2. Dependability worksheet with possible identified fault.

Fault classes		Fault subclasses	Software flaws	Physical interference	Mitigation
Persistence	XV	Transient		Lock communication protocol	Check the communication rate between the subsystems in the communication bus
Phase of creation or occurrence	I	Development	Traffic of unspecified information		Define operations to be performed in case of not receiving specified information

Source: Elaborated by the authors.

MODULE 5

The objective of this module is to validate the interoperability of real subsystems on the communication bus. This test was performed between the real OBC and the SLP. These subsystems were configured with mitigations for possible failures identified in previous tests. After implementing the mitigations, the tests met the interoperability requirements.

MODULE 6

In this module, the interoperability test between the communicating subsystems is performed using a FEM. This mechanism plays an important role in this module because the injections of faults are not intrusive in the communicating subsystems, testing the ability of each subsystem to withstand external faults.

The FEM acts as follows: i) it receives a message from the subsystem that is sending information to the other subsystem; ii) it injects a fault in this message; and iii) it retransmits this changed message to the target subsystem.

In this test module, performed on the NanosatC-BR2 satellite, the FEM was connected to the communication bus between the actual OBC subsystem and the simulated SLP subsystem. One of the tests carried out in the FEM was to inject a fault in the information sent by the OBC so that the information received by the SLP did not meet the specifications of the requirements.

This test allowed verifying that the injected fault had already been predicted and reported in the dependability worksheet and in the FMEA table during the evolution of the tests. It was identified that the necessary robustness had already been implemented in the SLP model during the previous tests.

This test allowed verifying that during the evolution of the tests, possible faults were identified, and the necessary mitigations were implemented to avoid interoperability failures among the communicating subsystems.

DISCUSSION

According to Jacklin (2015), a survey conducted with more than 165 articles available in the public domain revealed little use of software V&V approaches in the development of small satellites. The existing laboratories developed to support the assembly, integration, and testing (AIT) phases of large satellites have an established process to perform verification at the system level. In this phase, software embedded in the subsystems is considered validated, and the interoperability between SiSs is not stressed in the functional testing.

Usually, interoperability testing of a given SiS is covered by the V&V process adopted in the software engineering of that SiS, which makes use of simulators to represent the other SiS with which it interacts. Thus, it can be observed that the requirement specification is a critical baseline for the software implementation and the needed simulators as well. If two communicating SiSs have a different interpretation regarding a particular interoperability requirement, software faults will occur. Some examples of failures detected during the NanosatC-BR2 tests were:

- Connection issues on the I2C bus. This bus showed some connection failures, such as signal loss during some tests;
- Detection of storage and transmission limits of information packets depending on the board used. The Arduino UNO board has more limitations than the Mega board;
- Communication between subsystems can freeze due to gaps in the specification of the expected behavior of each SiS in certain unforeseen situations;
- Lack of a defined execution time for unforeseen situations.

The main contribution of the use of the SATS environment is to make suppliers understand the expected behavior of the SiS to be developed, from the perspective of interoperability. This early interoperability requirement verification through modeling execution and simulation avoids high-cost rework in the future. Simulation in the preliminary design phase is used to verify whether the interoperability requirements were modeled as specified. This approach makes it possible to detect and remove faults in the models early in the development cycle, when the real software has not yet been developed. This saves cost and time because interoperability failures usually occur late in the system integration phase.

Another contribution is the low-cost of the testing facility. Scalable Architecture Test System architecture is a viable alternative for low-cost nanosatellite projects because it reduces the effort in both managing test cases and reusing them through different phases, from MIL to HIL. This is achieved thanks to the application of the MBT and MDE approaches for test case generation and code generation, respectively.

However, the presented approach demands some extra effort. In particular, the complexity involved in the use of MBT and MDE tools should be highlighted. In general, there is a steep learning curve. It was also found that code generation is not always completely reliable: on some occasions, it became necessary to alter the generated code to meet the project requirements. It is expected that these problems will be mitigated as tools become more mature. Additionally, the creation of ad-hoc MBT/MDE tools, especially tailored to the nanosatellite domain, is a viable alternative.

Considerable effort was also directed toward the analysis of the possible faults presented in the dependability tree, as they require analysis by domain experts. The same applies to the elaboration of the information that will make up the dependability worksheet and the "faults and effects" table. This information must be analyzed by experts with knowledge of the identified classes of possible faults. Such information can be reused across different projects using the same nanosatellite family, thus improving the efficiency of the V&V process in this domain.

CONCLUSION

This paper addresses the problem of verifying and validating software embedded in CubeSat-based satellites. A systematic testing approach implemented through the use of a SATS is proposed for the V&V of interoperability requirements of communicating SiSs. Based on modeling and the expected behavior of the communicating SiSs, specified through interoperability requirements, test cases are generated from the model execution (MBT). The transformed models derive codes (MDE) that operate in simulated environments (MIL/SIL) through the real communication bus for requirements verification purposes and test case validation. These test cases will be reused later for requirements validation during the integration of the real SiSs.

MDE and MBSE approaches are extensively explored to support the preliminary design of SiSs and simulations in the development cycle of a satellite. These approaches address the modeling of space mission requirements and safety concepts, as presented in the related section. However, to the best of the author's knowledge, none of them combines the MBT and MDE approaches, along with the concepts of dependability, interoperability, and robustness. Besides enriching interoperability requirements with robustness aspects, the approach presented in this paper provides an improvement in identifying and avoiding potential space mission faults. This is achieved by using a dependability taxonomy (Avizienis *et al.* 2004) that supports systematic fault analysis and mitigation.

One of the main advantages of the proposed approach is the reuse of test sets by using the MBT and MDE approaches together, reducing the effort involved in transforming models and generating codes. Additionally, the use of the proposed dependability worksheet and "faults and effects" table provides another way to avoid possible faults, serving as a reference and historical database. The dependability worksheet covers the possible faults foreseen in the dependability taxonomy, allowing V&V through traceability of the potential sources and propagation of the faults that may occur.

Before conducting HIL testing, MIL simulation is carried out. The MIL allows checking for modeling changes to fulfill the requirements even before the software is developed.

The presented approach allows for reuse in different phases of the same mission, through test scenarios, which enables adding or removing subsystems of the satellite in the test loop. There is also the possibility that the same test system be used on different satellites of the same family. The laboratory infrastructure necessary to implement the approach is COTS, effective, low-cost and flexible enough to test several nanosatellite mission PLs that adopt the CubeSat standard.

As challenges for future works, it is envisioned that the use of the SATS approach will be used in the V&V of flight operation planning during the satellite operation phase in orbit.

In this scenario, SATS materializes the operational simulator role, supporting operational engineers in the task of assessing the behavior/effect of a telecommand sequence on the ground before it actually sent to the satellite.

CONFLICT OF INTEREST

Nothing to declare.

AUTHORS' CONTRIBUTION

Conceptualization: Conceição CAPL and Mattiello-Francisco MF; **Methodology:** Conceição CAPL and Mattiello-Francisco MF; **Writing - Original Draft:** Conceição CAPL and Mattiello-Francisco MF; **Final approval:** Conceição CAPL.

DATA AVAILABILITY STATEMENT

The data will be available upon request.

FUNDING

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
Finance Code 001.

ACKNOWLEDGMENTS

The authors would like to acknowledge the Programa de Pós-Doutoramento at INPE for supporting this research work.

REFERENCES

[ECSS] European Cooperation for Space Standardization (2018) ECSS-E-ST-10-02c Rev.1 – Verification [accessed Mar 25 2024]. https://ecss.nl/standard/ecss-e-st-10-02c-rev-1-verification-1-february-2018/

[INPE] Instituto Nacional de Pesquisas Espaciais (2015) Sistemas espaciais e tecnologia: documento de controle de interfaces do computador de bordo do satélite NanosatC-BR2. São José dos Campos: INPE.

Abrahão S, Bordeleau F, Cheng BHC, Kokaly S (2017) User experience for model-driven engineering: challenges and future directions. Paper presented 2017 International Conference on Model Driven Engineering Languages and Systems. IEEE; Austin, EUA. https://ieeexplore.ieee.org/document/8101269

Almeida DP (2016) Software C&DH embarcado em nanossatélites (ScdhNa). São José dos Campos: INPE.

Amason D (2008) SDO FlatSat facility. [accessed Oct 29 2023]. https://ntrs.nasa.gov/api/citations/20080023611/downloads/20080023611.pdf

Avizienis A, Laprie JC, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Depend Secure Comp 1(1):11-33. https://doi.org/10.1109/TDSC.2004.2

Barcellos JCE, Spengler AW, Seman LO, Silva RDC, Roldan HP, Bezerra EA (2023) FlatSat platforms for small satellites: a systematic mapping and classification. IEEE J MASS 4(2):186-198. https://doi.org/10.1109/JMASS.2023.3249044

Batista CLG, Martins E, Mattiello-Franscisco MF (2018) On the use of a failure emulator mechanism at nanosatellite subsystems integration tests. Paper presented 2018 IEEE 19th Latin American Test Symposium. IEEE; São Paulo, Brazil. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8347242

Binder R (2002) Testing object-oriented systems: models, patterns and tools. Boston: Addison-Wesley.

Broy M, Jonsson B, Katoen JP, Pretschner A (2005) Model-based testing of reactive systems. Berlin: Springer.

Ciacchella G, Mohammad A, Zurawski (2024) An open-source method for model-based development of embedded systems: experience report from a CubeSat student project. Paper presented 2024 74th International Astronautical Congress. IAC; Baku, Azerbaijan. https://hal.science/hal-04564986v1/file/IAC-23%2CD1%2CIP%2C6%2Cx77646.paper-ext.300dpi.v1-5.pdf

Desmoulin A, Viho C (2007) Automatic interoperability test case generation based on formal definitions. https://www.irisa.fr/armor/lesmembres/Desmoulin/desmoulin-viho-fmics07-vf.pdf

Duarte RO, Vale SRC, Martins-Filho LS, Torres FE (2020) Development of an autonomous redundant attitude determination system for Cubesats. J Aerosp Technol Manag 12:e3020. https://doi.org/10.5028/jatm.v12.1166

Helvajian H, Janson SW (2008) Small satellites: past, present and future. Chantilly: Aerospace Press.

Hestad T, Barabash V, Laufer R (2023) The APTAS student CubeSat mission: a case study for reflective practitioner in education and student teams: 2. Adv Space Res 72(6): 2245-2258. https://doi.org/10.1016/j.asr.2023.06.024

Itemis AG (2024) Yakindu. [accessed Jun 15 2023]. https://updates.yakindu.org/statecharts/products/standard/latest/

Jacklin SA (2015) Survey of verification and validation techniques for small satellite software development. Washington, D.C.: NASA.

Kanavouras K (2021) Design of fault detection, isolation and recovery in the CubeSAT nanosatellite (doctoral dissertation). Thessaloniki: Aristotle University of Thessaloniki.

Langer M, Boumeester J (2016) Reliability of CubeSats: statistical data, developers' beliefs and the way forward. Paper presented 2016 30th AIAA/USU Conference on Small Satellites. AIAA; Logan, USA.

Mattiello-Francisco MF, Martins E, Cavalli AR, Yanod ET (2012) InRob: an approach for testing interoperability and robustness of real-time embedded software. J Syst Softw 85(1):3-15. https://doi.org/10.1016/j.jss.2011.02.034

Myers GJ, Sandler C, Badgett T (2011) The art of software testing. Hoboken: John Wiley & Sons.

Petrenko AK, Schlingloff H (2013) Eighth workshop on model-based testing. Computer Science 111. https://doi.org/10.4204/EPTCS.111

Rabasa GO (2015) Methods for dependability analysis of small satellite missions (doctoral dissertation). Torino: Politecnico di Torino.

Rizza A, Piccolo F, Pugliatti M, Panicuccu P, Topputo F (2022) Hardware-in-the-loop simulation framework for CubeSats proximity operations: application to the Milani mission. Paper presented 2022 3rd International Astronautical Congress. IAF; Paris, France. https://re.public.polimi.it/retrieve/75eb68ea-33c1-47d3-9fc1-dca0e7de3cad/RIZZA01-22.pdf

Schmidt DC (2006) Model-driven engineering. IEEE Comp 39(2):25-31.

Schuch N, Durão O, Silva M, Mattiello-Francisco F, Silva A (2017) NanosatC-BR STATUS: a joint CubeSat: BASED program developed by INPE and UFSM. [accessed Jun 27 2020]. https://www.researchgate.net/publication/322746790_NANOSATC-BR_STATUS_-_A_JOINT_CUBESAT-BASED_PROGRAM_DEVELOPED_BY_INPE_AND_UFSM

Schuch NJ, Marques RP, Lanza I, Pedros FS, Mantovani LQ, Silva MR, Batista CLG, Almeida DP, Mattiello-Francisco MF, Durao OSC, et al. (2022) The NANOSATC-BR, CubeSats development program: capacity building with the NANOSATC-BR2's launch, first operations and expected results through INPE's ground station at Santa Maria. Braz J Geophys 40(3):1-18. https://doi.org/10.22564/brjg.v40i3.2177

Silva MJ (2024) Análise comparativa entre os processos de desenvolvimento de software de OBDH do NANOSATC-BR2 e do AMAZONIA-1 (master's thesis). São José dos Campos: Instituto Nacional de Pesquisas Espaciais. Available at: http://mtc-m21d.sid.inpe.br/col/sid.inpe.br/mtc-m21d/2024/07.26.12.14/doc/publicacao.pdf

Souza SCM, Carvalho RA (2021) Automated driver testing for small footprint embedded systems. [accessed Aug 14 2023]. https://www.researchgate.net/publication/351342673_Automated_Driver_Testing_for_Small_Footprint_Embedded_Systems

Srinivas N, Panditi N, Schmidt S, Garrelfs R (2014) MIL/SIL/PIL approach a new paradigm in model based development. [accessed Nov 15 2023]. https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/automotive/files/in-expo-2014/mil-sil-pil-a-new-paradigm-in-model-based-development.pdf

Swartwout M (2015) Secondary spacecraft in 2015: analyzing success and failure. Paper presented 2015 IEEE Aerospace Conference. IEEE; Big Sky, USA. https://ieeexplore.ieee.org/abstract/document/7119175

Swartwout M (2016) Secondary spacecraft in 2016: why some succeed (and too many do not). Paper presented 2016 IEEE Aerospace Conference. IEEE, Big Sky, USA. https://ieeexplore.ieee.org/document/7500791

Swartwout M (2019) CubeSat mission success: are we getting better? [accessed Sep 20 2023]. http://mstl.atl.calpoly.edu/~workshop/archive/2019/Spring/Day%201/Session%202/MichaelSwartwout.pdf

Uppaala University (2024) Uppaal version 4.1.19. [accessed Jun 16 2023]. https://www2.it.uu.se/research/group/darts/uppaal/download.shtml

Yost B, Weston S (2024) State-of-the art small spacecraft technology. Washington, D.C.: NASA.