CubeSat Flight Software: Insights and a Case Study

Mohammed Eshaq[*], M. Sami Zitouni[†], and Shadi Atalla[‡]
University of Dubai, Dubai, P. O. Box: 14143, United Arab Emirates

Saeed Al-Mansoori[§]
Mohammed Bin Rashid Space Center (MBRSC), Dubai, P. O. Box: 211833, United Arab Emirates

Malcolm Macdonald[**]
University of Strathclyde, Glasgow, G1 1XQ, United Kingdom

The development of efficient flight software (FSW) for CubeSats faces significant challenges due to lack of mainstream guidelines and frameworks. Addressing the challenge of developing such FSW within the ever evolving yet stringent hardware constraints forms the core of this study. This paper presents a comprehensive analysis of FSW specifications, development challenges, and proposes a novel approach to software design that maximizes functionality while adhering to hardware limitations. It begins by scrutinizing current trends in CubeSat FSW, emphasizing the significance of modularity and reusability for successful, adaptable, and repeatable space missions. Drawing from a diverse array of literature sources, various aspects of CubeSat FSW is explored, encompassing design methodologies, subsystems, mission applications, reliability, fault tolerance, testing, validation, and communication. Subsequently, a case study is introduced featuring an app-based FSW solution tailored for a 12U CubeSat with a 5G Internet-of-Things payload and green propulsion. This case study incorporates insights from the literature review, featuring a service-oriented. The FSW solution includes a user-friendly command line interface for efficient onboard and ground communication, a script engine for timely in-orbit execution and payload control, and a bootloader for in-mission updates, enhancing adaptability and future readiness. The analysis and case study reaffirm the pivotal role of robust and efficient flight software in mission

[*] Corresponding Author (meshaq@ud.ac.ae), Senior Embedded Engineer, Mohammed Bin Rashid Space Center (MBRSC) Lab, College of Engineering & IT.
[†] Assistant Professor, Mohammed Bin Rashid Space Center (MBRSC) Lab, College of Engineering & IT.
[‡] Associate Professor, Mohammed Bin Rashid Space Center (MBRSC) Lab, College of Engineering & IT.
[§] Director, Remote Sensing Department.
[**] Professor and Director, Applied Space Technology Laboratory (ApSTL), Department of Electronic and Electrical Engineering.

This is a peer-reviewed, accepted manuscript of the following paper: Eshaq, M., Zitouni, M. S., Atalla, S., Al-Mansoori, S., & Macdonald, M. (in press). CubeSat flight software: insights and a case study. Journal of Spacecraft and Rockets.

success, while underscoring the need for freely available, modular, and reusable solutions to foster innovation in the field, ultimately reducing reliance on commercial products or continual redevelopment.

I. Introduction

TECHNOLOGICAL advancements have led to the creation of smaller satellites that can be easily held and handled by a single person. A popular category of such smaller satellites is the nanosatellite category (weighing between 1 to 10 KG) [1]. An even more popular type of nanosatellite is the Cube Satellite (CubeSat), a miniaturized satellite built out of standardized cubes; the name comes from the cubed units used to build the satellite. Any CubeSat may have the size of 1 Unit (1U), or multiples of such units where each unit measures 10 x 10 x 10 cm. These units typically have a little over 1 kilogram mass per unit. CubeSats were first developed in 1999 by California Polytechnic State University and Stanford University and have since grown in popularity due to their low cost and flexibility [2]. A CubeSat can be quickly built with in-house-developed or commercial-off-the-shelf (COTS) electronics and components. CubeSats are typically used for space research and commercial use [3]. This means that developers with limited to no experience in satellite technology away from large space agencies, such as students at universities and research centers or even enthusiasts, can now explore the realms of space with CubeSats [4]. CubeSat hardware standardization and adoption in academia and industry has led to increased volumes of production of off-the-shelf components. Although this has resulted in a significant reduction in satellite development time and cost, a considerable amount of mission development time and effort is still spent on flight software (FSW) development [5] unless the code is reusable [6].

In exploring the domain of CubeSat flight software (FSW), this study first undertakes a comprehensive literature review to scrutinize existing FSW development practices, design methodologies, and the integration challenges inherent to CubeSat missions. Through this detailed examination, we identify critical shortcomings in the current approaches, particularly in terms of modularity, reusability, and efficiency. These shortcomings highlight the need for a more streamlined framework that can adapt to the rapid advancements in space technology and mission requirements.

The analysis of existing Flight Software (FSW) developments reveals a notable absence of several key features, deemed critical for the efficacy and success of CubeSat missions. These missing elements, and their impacts on both the FSW and the overarching mission objectives, are identified and summarized as shown in Table 1.

Table 1 Missing Essential Features in Existing Solutions and Their Impact on Mission Succes

Essential Features Often Lacking in Current FSW Solutions	Impact on FSW and the Mission
Modularity and Service-Oriented Architecture	Constrains the potential for software reuse in subsequent missions, leading to increased developmental efforts and costs.
Multi-Layered Architecture	Diminishes software abstraction and portability, adversely affecting modularity and the ability to adapt software across different mission contexts.
Utilization of FreeRTOS	The preference for more sophisticated operating systems necessitates the deployment of more robust hardware, potentially escalating mission costs and complexity.
Implementation of a Command Line Interface	Complicates testing procedures and the development of efficient script engines, potentially hindering operational agility.
Incorporation of a Script Engine	Restricts the versatility of Mission Control Systems for use in future missions, limiting the longevity and adaptability of the software.
Integration of a Bootloader	Absence of in-mission software update or patching capabilities renders any post-launch software issues intractable, risking mission failure.

This work's significant contribution emerges in the form of a case study detailing the development and deployment of a novel FSW framework for a 12U CubeSat mission. This framework, characterized by its service-oriented and multi-layered architecture, introduces a modular app-based software and runs on FreeRTOS. The solution features an intuitive command-line interface, a versatile script engine for enhanced in-orbit operations, and a bootloader for in-mission FSW updates. The case study not only serves as a practical implementation of the proposed framework but also validates its efficacy in improving mission outcomes, offering a scalable and adaptable solution for future CubeSat projects.

While widely adopted frameworks like NASA's Core Flight System (cFS) [7] and F Prime [8] provide strong modularity and service-oriented architecture, they have notable limitations. For instance, cFS does not natively support FreeRTOS, a lightweight, widely used real-time operating system ideal for resource-constrained CubeSat missions. Additionally, neither cFS nor F Prime provide a built-in script engine for automating satellite operations, or a flexible bootloader for in-mission software updates. These features are essential for mission adaptability and efficiency. They have been specifically addressed in this work to improve scalability, simplify operations, and enhance flexibility during the mission.

By bridging the gap between theoretical research and practical application, this study provides a robust foundation for advancing CubeSat FSW development. The combination of a thorough literature review with a space-proven case study underscores the dual focus on understanding the current landscape and contributing a tangible, innovative framework that addresses identified deficiencies. This approach aims to foster broader participation in CubeSat development, encouraging open-source collaborations and setting a new benchmark for mission success.

The paper begins with an extensive literature review, comprehensively evaluating advancements and challenges in CubeSat FSW. Subsequently, the paper transitions into a detailed case study, focusing on the FSW design and implementation development for the PHI-Demo spacecraft. The CubeSat is a 12U CubeSat featuring a 5G Internet-of-Things (IoT) subsystem as its primary payload and a green propulsion system as a secondary, demonstrative payload. This case study not only illustrates insights gleaned from the literature review but also serves as a practical blueprint for future software development endeavors.[††]

A. Challenges and Motivation

The National Academies' report titled "Achieving Science with CubeSats: Thinking Inside the Box" [2] provided an overview of CubeSats developed to facilitate cost-effective access to space for scientific and technological purposes. The authors asserted that CubeSats often have lower reliability than traditional satellites. This claim can be validated by examining Fig. 1 as it shows the status of all CubeSat launched, and scheduled to launch, to date and their mission status [9]. In the graph, the mission statuses are defined as:

1. **Pre-launch:** CubeSats that are in the planning, design, or manufacturing stage and have not yet been launched into space.
2. **Launch Fail:** CubeSats that experienced a failure during the launch process, resulting in an unsuccessful mission.
3. **DOA (Dead on Arrival):** CubeSats that, upon reaching orbit, were found to be non-functional or failed to establish communication with ground stations.
4. **Early Loss:** CubeSats that became non-functional or lost communication shortly after the launch, before completing their intended mission objectives.
5. **Partial Mission:** CubeSats that accomplished some, but not all, of their mission objectives before encountering issues or reaching the end of their operational lifetime.
6. **Full Mission:** CubeSats that successfully completed all their intended mission objectives and demonstrated full operational capability.
7. **Unknown:** CubeSats for which the mission status is uncertain or has not been reported, making it difficult to determine their success or failure.

[††] The case study section is an expanded version of our previously published conference paper [40].

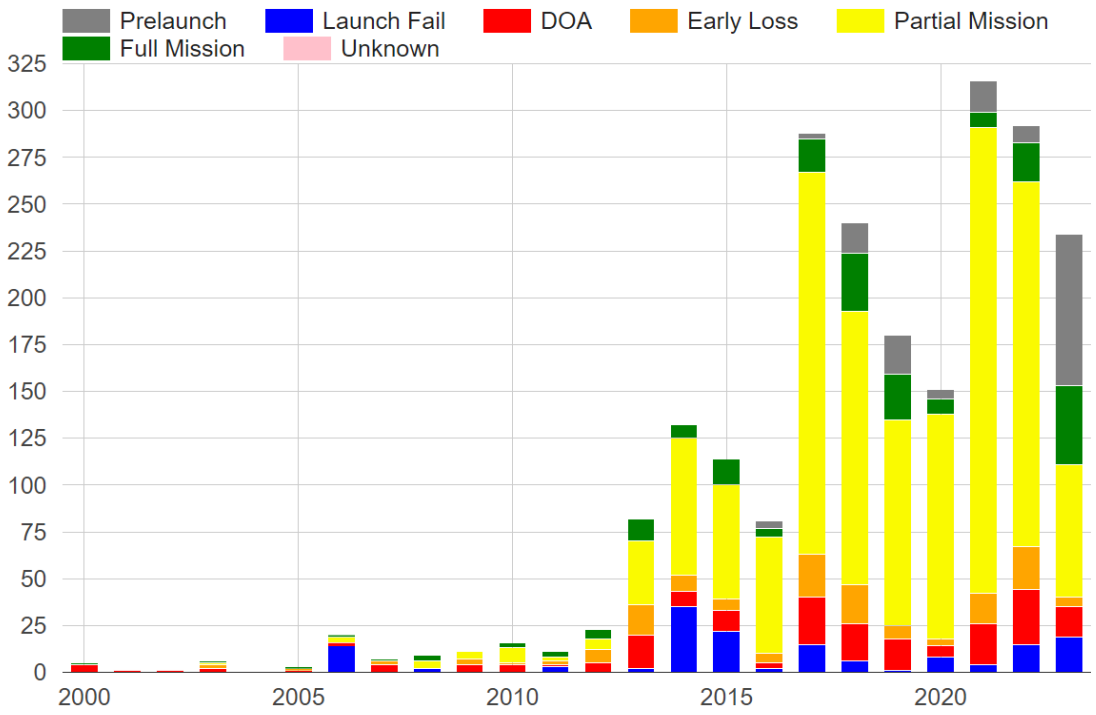

Fig. 1 CubeSat Mission Status per Year [9]

One way to mitigate the risk of failure is to adopt a ‘fly-learn-re-fly’ approach [2], where two flight models are developed, and the second flight model is modified and launched if any issues arise during the first flight. This approach can decrease development time, and testing costs, but can also result in increased costs, especially when the cost of a launch is considered. Additionally, this approach can lead to delays in achieving mission objectives if issues arise during the first flight that requires significant modifications to the second model. Despite these potential drawbacks, the ‘fly-learn-re-fly’ approach has been widely adopted for CubeSat development, due to its effectiveness in improving mission success rates. Nevertheless, other sources, in section B, suggest that FSW is potentially the most critical aspect in determining mission success for CubeSats.

In light of the challenges highlighted by the National Academies' report and the diverse mission outcomes of CubeSats as shown in Fig. 1, this paper is motivated by the pressing need to enhance CubeSat mission reliability and success rates. Despite the advances in CubeSat technology, the variance in mission outcomes—from early loss to full mission success—underscores a critical gap in current CubeSat development practices, particularly in the realm of flight software (FSW). The 'fly-learn-re-fly' approach, while effective, points to deeper underlying issues in the design

and testing phases that could benefit from improved FSW methodologies. Consequently, this paper aims to address these gaps by presenting a comprehensive analysis and a novel framework for FSW development, tailored to mitigate the risks identified and to streamline the process for achieving mission objectives. Our motivation is rooted in the belief that through targeted improvements in FSW, CubeSats can achieve higher reliability and operational success, making space more accessible for scientific and technological exploration.

B. The Significance of FSW

Just as with any other satellite, a CubeSat comprises multiple subsystems each with a specific function: power, communication, attitude control, and so forth. These subsystems are embedded systems connected to the Onboard Computer (OBC). The OBC is responsible for processing commands and responses, managing communication with the ground, and overseeing the overall operation of the satellite. Therefore, designing and implementing the FSW that runs on the OBC is vital to CubeSat missions.

CubeSat FSW development often significantly differs from that of large missions. The teams involved in CubeSat development are smaller, more open to adopting new technology, rely less on formal verification methods, and often produce FSW solutions with only a few thousand lines of code compared to millions of lines in large flight projects [2].

The FSW must provide “numerous services, such as computer boot-up and initialization, time management, hardware interface control, command processing, telemetry processing, data storage management, FSW patch and load, and fault protection” [10]. Implementing FSW is more challenging than other commercial embedded software; once the satellite has launched, there is no interaction with users unless the satellite makes contact with the ground station. Even then, it can only communicate via uplink and downlink during short, intermittent time windows. Most of the mission time, the FSW runs unsupervised by the ground. Thus, it must be able to recover from faults on its own [10]. To achieve this and despite all the recent advancements in technology, a significant portion of mission development resources are allocated to the development of FSW [5].

Many sources suggested that FSW is a critical aspect in determining mission success for CubeSats. The work in [11] emphasized the importance of the FSW. Gonzalez et al. [4] emphasized the importance of FSW architecture and monitoring during the software life cycle to ensure software quality and reduce mission risk. Furthermore, [12] asserts that CubeSats do not have an outstanding record of mission success. Therefore, the authors provided recommendations to improve the likelihood of mission success for future CubeSat development projects, including implementing

common-sense practices, such as thorough testing and mission assurance. Overall, these papers suggested that FSW is critical in determining mission success for CubeSats and should be carefully designed, monitored, and tested to ensure software quality and reduce mission risk. Yet, the National Academies report [2] highlighted that CubeSat Flight Software (FSW) development lagged behind hardware advancements in the CubeSat field.

C. Hardware Components: Typical and Emerging

Understanding the composition of a CubeSat requires a detailed look at its hardware components. This section will summarize the typical components and highlight emerging technologies pertinent to CubeSat hardware.

1) **OBC:** As the primary computer, it governs mission operations, communication, and overall satellite management, with the FSW operating on it. The in-depth discussion on this component will be the focus of the upcoming sections.
2) **Electrical Power Subsystem (EPS):** This system collects energy from solar panel arrays, storing it in the satellite's batteries. With CubeSat growth, power generation through deployable solar arrays has become more prevalent. Future advancements in solar cell technology could potentially revolutionize CubeSat power generation. CubeSats typically use Lithium-ion and Lithium-polymer batteries due to their efficiency and affordability [2].
3) **Communication Subsystem (COM):** Essential for ground communication, CubeSats commonly employ UHF, VHF, L-band, X-band, Ku-band, or Ka-band frequencies. Technological boundaries of CubeSat communication are continually pushed to accommodate more sophisticated missions, with inter-satellite communication links now becoming more common.
4) **Attitude Determination and Control Subsystem (ADCS):** Determines the satellite's position and orientation, executing required maneuvers. Modern CubeSats have moved beyond passive control systems and now integrate advanced techniques and systems like Sun and Earth sensors, angular rate sensors, star trackers, and reaction wheels. Some CubeSats achieve an accuracy of less than 10 arcseconds. Furthermore, GPS is now being widely used for orbit determination, [2] although CubeSats are considered physically too small to use GPS for attitude control.
5) **Payload Subsystems:** These are mission-specific subsystems delivered to space to fulfill mission objectives. They might include imaging instruments, specialized communication devices, or emerging technologies for space-testing.

Advancements in technology also introduce novel components and subsystems to CubeSats, such as:

1) **Advanced Payloads**: Sophisticated payloads, such as high-resolution cameras [13], thermal imaging [14], synthetic aperture radar (SAR) [15], hyperspectral imagers [16], and scientific instruments for Earth observation [17], space weather monitoring [18], and even deep space exploration [19] are increasingly being incorporated in CubeSats.
2) **Software-Defined Radio (SDR):** Offering reconfigurable communication systems, SDRs adds more flexibility to CubeSat missions. Several commercial off-the-shelf SDR subsystems are readily available [20,21].
3) **Optical Communication:** An alternative to radio frequency communication, optical communication systems allow larger data volumes or high-resolution imagery transmission [22]. Recent developments include low-power optical communications, providing potentially gigabits-per-second data rates [2], and even the use of LEDs rather than lasers to reduce both power and attitude control requirements [23,24].
4) **Propulsion Subsystem:** Small-scale propulsion systems are being developed for CubeSats [25], enabling more complex maneuvers, orbit adjustments, and extended mission durations, as in the case study of this work, and deorbiting at end-of-life.
5) **Active Thermal Control**: While passive thermal control methods like paint, thermal tape, and Multi-Layer Insulation (MLI) are still used. CubeSats now also incorporate active thermal control systems, such as heaters and thermal switches, to maintain temperature stability [2]. The CubeSat in the presented case study incorporates this feature.
6) **Deployable Structures:** CubeSats are increasingly using deployable structures and mechanisms, such as extendable solar panels, antennas, booms, and de-orbit devices, to enhance their capabilities, improve power generation, increase communication range, and facilitate end-of-life disposal [2].
7) **Onboard Artificial Intelligence and Machine Learning:** Advanced data processing and embedded vision capabilities are being integrated into CubeSats, enabling real-time data analysis, autonomous decision-making, and optimization of mission objectives. A study in [26] showed simulated results for such an approach in a 6U CubeSat.

8) **Swarm, Cluster, or Constellation Technologies:** CubeSats are increasingly being deployed in groups to achieve better coverage, data redundancy, or coordinated measurements. These arrangements demand unique architecture and internal components compared to traditional stand-alone CubeSats.

The rest of the paper is organized as follows: first, a comprehensive literature review is conducted. The references are classified and a summary of each is provided. Then, the state-of-the-art is analyzed and characteristics for modular FSW are drawn. After that, a Case Study of a 12U CubeSat is presented. The FSW Design and Architecture for this CubeSat is also showcased and tested. Finally, discussions and conclusions are drawn.

II. Literature Review

A comprehensive search was conducted on the Scopus database and Google Scholar using the keywords “Flight Software” and “CubeSat.” This search strategy aimed to identify a wide range of papers specifically addressing the intersection of FSW and CubeSat technologies to help build the CubeSat FSW. The inclusion criteria were that the references must specifically address FSW in CubeSats. Following the retrieval of a total of 63 papers, they were carefully reviewed and categorized into distinct thematic areas, namely ‘FSW Design and Development’, ‘CubeSat Subsystems and Components’, ‘CubeSat Missions and Applications’, ‘CubeSat Reliability, and Fault Tolerance and Anomaly Analysis’, ‘CubeSat Testing and Validation’, and ‘CubeSat Communication and Networking’. Some papers were relevant to multiple categories, yet they were categorized under the category that best aligned with the paper’s primary focus. This categorization facilitated a structured analysis of the current state of research in FSW for CubeSats, enabling the identification of key trends, challenges, and future directions in this rapidly evolving field. Fig. 2 below shows how these topics were categorized, while Fig. 3 shows the number of papers in each category.

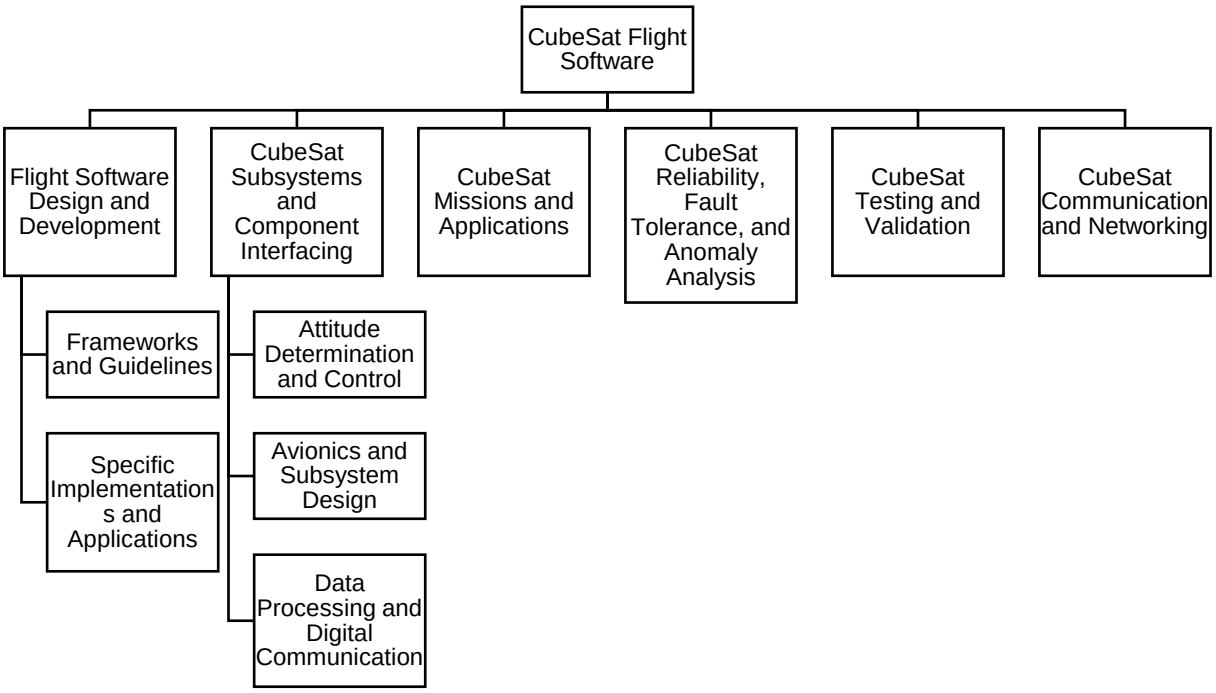

Fig. 2 CubeSat FSW Taxonomy

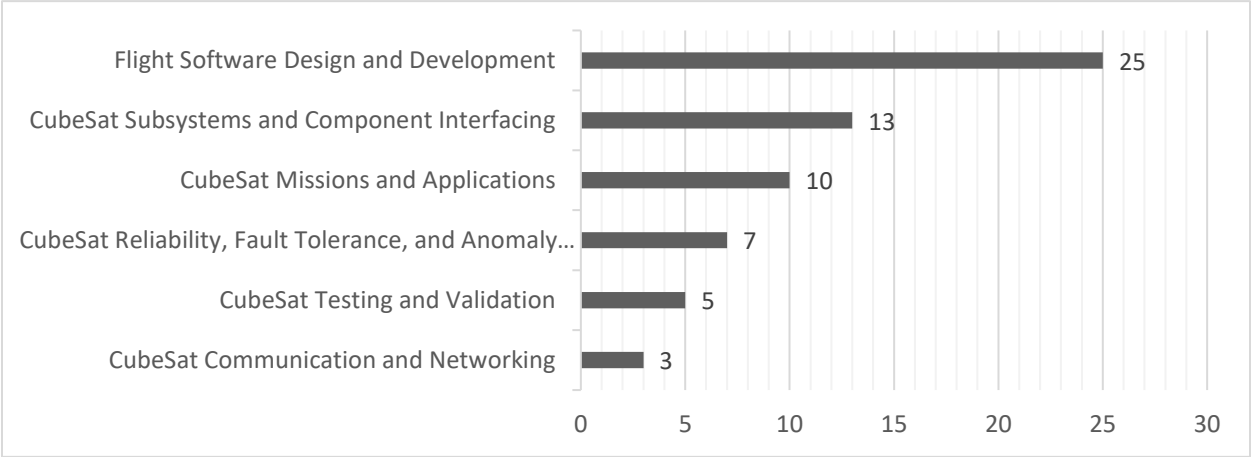

Fig. 3 Number of Papers in each Category

Furthermore, the papers were grouped based on publication year as shown in Fig. 4. The oldest paper found was dated 2006, and the newest was 2023.

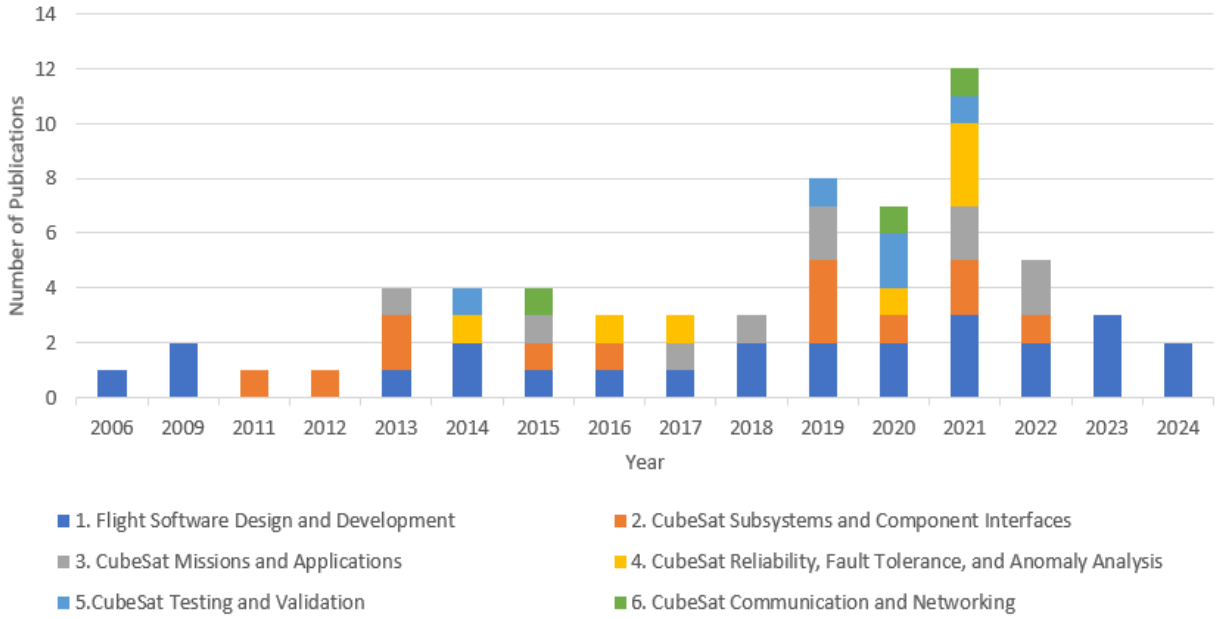

Fig. 4 Number of Publications per Year

Based on Fig. 4, the following can immediately be concluded. The number of publications in the field of CubeSat FSW has generally increased over time, suggesting that the field has gained more interest and attention in recent years. This could however simply be attributed to technological advancements and increased interest in space exploration and research. The highest number of publications occurred in 2021, with 12 publications, followed by slight drop in the number of publications in 2022. Between 2013 and 2020, there was a relatively steady growth in the number of publications per year, indicating a consistent interest in CubeSat FSW during this period. This growth appears to lag the growth in CubeSat missions seen in Fig. 1.

A. FSW Design and Development

This category addresses the design and development aspect and can be further divided into ‘Frameworks and Guidelines’ and ‘Specific Implementations and Applications’. They cover methodologies, principles, and practical examples for creating FSW for various projects and missions.

1. Frameworks and Guidelines

To pave the road for FSW development, many attempts were made to provide frameworks and guidelines for such an undertaking. For example, [5] introduced a highly modular software framework for nanosatellites to reduce software development time, while [27] explored the design and implementation of reusable and reliable flight-control

software for cost-efficient CubeSat missions. On the other hand, [28] presented design guidelines for general-purpose payload-oriented nanosatellite software architectures, focusing on improving system robustness, modularity, and autonomy. Nevertheless, [4] proposed an architecture-tracking approach to evaluate modular and extensible FSW for CubeSat nanosatellites, while [29] discussed the development of an FSW framework specifically for student CubeSat missions. Furthermore, [30] explored design techniques for onboard software of nanosatellites, focusing on the FSW of ICUBE-1 CubeSat from Pakistan. Also, a publication that can be used as a guideline, [31], described the design and implementation of generic FSW for a CubeSat to increase reusability and reduce future development costs. The paper in [32] introduces a model-based systems engineering (MBSE) methodology combined with documentation as code (docs-as-code) to streamline the CubeSat flight software development process.

An open-source pysimCoder-based framework [33], and a recent study in [34] both present innovative approaches to CubeSat flight software development, specifically designed to support academic projects. The pysimCoder-based work allows for dynamic system modeling, simulation, and real-time code generation from block diagrams, streamlining the development cycle. Meanwhile, the modular architecture introduced in [34] employs agile software development methodology and promises enhancing software efficiency and scalability.

In [10], the complexity of FSW in NASA's space missions was investigated, with the authors providing recommendations for managing and reducing complexity. The article in [35] discussed the implementation of NASA's open-source cFS framework in an undergraduate space mission program in Mexico. Furthermore, [8] introduced F Prime, an open-source framework by a team from NASA for small-scale FSW systems. Finally, the paper in [36] presented a novel approach to CubeSat FSW development using the Robot Operating System (ROS).

2. Specific Implementations and Applications

The papers selected for this category discussed various examples of FSW development for particular applications. For example, [11] and [37] focused on specific CubeSat missions, EIRSAT-1 and KySat-1, respectively, and both discussed the challenges while developing and testing FSW in an academic environment. The authors in [38] presented an analysis of software architectures for deep-space navigation filters and argue for a modular approach supported by the Basilisk astrodynamics framework. Similarly, [6,39,40] emphasized the importance of software reuse, modularity, and scalability in FSW development. Furthermore, an older article, [1], described the design and implementation of the bootloader and COM for the CanX-2 nanosatellite, while [3,40] presented a multi-layered architecture for a CubeSat mission software running on an STM32-based OBC. Additionally, Fitzpatrick et al. in [41] explored the flight

software architecture for the SWARM-EX CubeSat mission, focusing on low-power design choices and novel communication protocols for spacecraft swarming, thereby illustrating another approach towards enhancing CubeSat mission efficacy with student engagement in mind.

Finally, [42] discussed the implementation of an embedded RTOS for the twin nanosatellite STUDSAT-2, while [43] discussed the challenges of over-the-air firmware updates for educational CubeSats. In summary, these articles highlighted the importance of robust, modular, and reusable FSW for small satellite missions while addressing the challenges and constraints faced mostly by university-based teams and low-resource satellite platforms.

B. CubeSat Subsystems and Component Interfacing

This category focuses on the essential building blocks that enable CubeSats to function effectively. The subsequent three subcategories, ‘Attitude Determination and Control’, ‘Avionics and Subsystem Design’, and ‘Data Processing and Digital Communication’, delve deeper into the intricacies of these subsystems, exploring their specific roles in maintaining the performance, stability, and data transmission capabilities of CubeSats.

1. Attitude Determination and Control

This section addresses various aspects of CubeSat Guidance, Navigation, and Control (GNC) systems, with some references focusing on attitude determination and control approaches. Other references emphasized the development of low-cost subsystems. For example, research by Gatherer and Manchester [44] proposed a magnetorquer-only attitude control technique that demonstrated improved performance over previous methods by using trajectory optimization. The article in [45] presented the ADCS for the Microwave Radiometer Technology Acceleration (MiRaTA) mission, which requires accurate and agile pointing. Moreover, [46] outlined the development and validation process of a highly automated GNC subsystem for on-orbit inspection applications, enabling proximity operations without human intervention. Reference [47] discussed a low-cost magnetic cleanliness routine for CubeSats, aiming to minimize disturbances during design and assembly while providing a spacecraft-specific disturbance model. Papers [48,49] emphasized rapid and low-cost development approaches, with [48] detailing the expedited GNC development schedule of Seeker, a free-flying inspector CubeSat, COTS sensors and lean development practices. Paper [49] focused on a low-cost star tracker, significantly reducing costs compared to commercial offerings while maintaining high performance. The star tracker uses COTS components and a modular design, making it compatible with various hardware configurations and FSW architectures.

Overall, the papers presented in this section showcased diverse GNC systems for CubeSats, addressing challenges in attitude determination and control, automation, cost reduction, and rapid development by employing various techniques, such as trajectory optimization, magnetic cleanliness routines, and the use of COTS components.

2. Avionics and Subsystem Design

The papers in this subcategory discussed various aspects of CubeSat design, development, and lessons learned from different missions. For example, [50] presented an analysis of two generations of avionics design for CubeSats, highlighting design principles relevant to other missions. Furthermore, [51] described the design, implementation, and testing of the subsystems in ICUBE-1 CubeSat, emphasizing software architecture and redundancy techniques to ensure reliability in orbit. Another work, [52], detailed the system design of the INCA spacecraft, a student-built CubeSat meant to demonstrate the functionality of a new Scintillator-SiPM-based neutron detector, covering various subsystems and lessons learned during assembly and integration. Lastly, the authors in [53] recounted the experiences and lessons learned from the UNITE CubeSat project, including challenges faced during the design, build, integration, test, delivery, and early operational phases, with a focus on the importance of communication, documentation, and testing for a successful mission.

3. Data Processing and Digital Communication

The publications in this category explored different aspects of information processing, digital communications, and data handling that occur among the subsystems in a satellite. For example, [54] implemented CubeSat Space Protocol (CSP) over Controller Area Network (CAN) and CSP over Serial Peripheral Interface (SPI) in a modular satellite system. Similarly, [55] discussed the heart unit of the ISTNanosat-1, a CubeSat developed to study the flyby anomaly phenomenon, emphasizing the unit's digital communications and command & data handling subsystems. Meanwhile, [56] presented a distributed computing architecture to address power constraints in a CubeSat with precision three-axis attitude control. It detailed the distributed design of operating modes and telemetry budget across two computing platforms, including a low-power flight computer and a high-speed auxiliary processor. While all papers investigate communication and data handling systems in CubeSats, their specific contexts and objectives differ, revealing diverse solutions and architectures.

C. Missions and Applications

The publications belonging to this category exhibit a diverse range of CubeSat missions and applications, addressing various challenges and design requirements. For example, the work in [57] built a CubeSat (3CAT-2) that

is focused on multi-constellation Global Navigation Satellite System Reflectometry (GNSS-R) and Global Navigation Satellite System Radio Occultation (RO GNSS-RO) experimental missions with ADCS and FSW validation campaigns to be conducted at Polytechnic University of Catalonia. Moreover, the work in [58] aimed to mature technologies, such as telescopes, while investigating space weather effects on instrument behavior. LightSail 2 technology [59] demonstrated controlled solar sailing in Earth orbit using a CubeSat platform, successfully harnessing momentum from solar photons.

IonSat [60] was a 6U CubeSat equipped with an ion thruster to maintain altitude in Very Low Earth Orbit (VLEO), using a step-down descent mission strategy. Another CubeSat was developed that is capable of High-Resolution Image and Video (HIREV) [61]. It explored the development of a 6U CubeSat platform with domestically manufactured parts. In another work, Near-Earth Asteroid (NEA) Scout CubeSat [62] focused on software techniques to address limitations in a small form factor, with technologies like target acquisition and onboard image calibration. IDEASSat [63] was a 3U CubeSat designed to measure ionospheric irregularities, while nSight-1 [64] was built to gather scientific measurements for the QB50 project and capture Earth observation images with the Gecko camera.

A system engineering approach for CubeSat design was proposed in [65] to ensure reliability, traceability, and reusability. Lastly, the book chapter in [66] demonstrated an interplanetary CubeSat mission accompanying the InSight mission to Mars, relaying Entry, Descent, and Landing (EDL) data to Earth during the process. These publications highlight the versatility and potential of CubeSats, with various missions and applications targeting Earth observation, interplanetary exploration, and technology demonstrations.

D. Reliability, Fault Tolerance, and Anomaly Analysis

The papers explore various aspects of CubeSat FSW in this category, focusing on reliability, fault tolerance, and autonomy. The works in [67] and [68] both emphasized the development of cost-efficient and reliable CubeSat architectures. While the former discussed the design of a reusable and fail-safe software architecture for student-built CubeSat missions, the latter presented the Southwest Low-Earh Orbit (LEO) EXplorer (SLX-6), a high-reliability CubeSat bus. Regarding fault tolerance, the studies in [69] and [70] introduced strategies to ensure CubeSat resilience against adversarial space environments. The earlier employed a deterministic state machine and fault-tolerant global memory structure, while the latter suggested a comprehensive, multi-level fault protection system for nanosatellites. Both [71] and [72] discussed techniques for reliability assessment; the former employed high-order Markov chains for CubeSat software, while the latter demonstrated onboard Model-Based Fault Diagnosis (MBFD) for CubeSat

ADCS using flight data. Lastly, [73] proposed a reference architecture for integrating autonomy-enabling components into mature FSW, allowing for more responsive and autonomous small satellite operations.

E. Testing and Validation

The articles in [74,75] focused on hardware-in-the-loop (HIL) testing for CubeSat missions, while the three publications in [76–78] discussed various software testing methodologies for CubeSat FSW. For example, [74] detailed the development of a system-level HIL test for the DICE mission, emphasizing the importance of simulating orbital dynamics, attitude dynamics, and environmental physics for testing subsystem interactions with the ADCS. Similarly, the authors of [75] reported their experience with HIL and software-in-the-loop (SIL) tests in the development of the MOVE-II CubeSat, emphasizing the benefits of including the electrical domain of the satellite for accurate power budget verification. The authors also described how the simulation environment was used to analyze issues detected after launch and verify the performance of new software developed to address in-flight anomalies.

Papers [76–78] focused on testing and mitigating software-related issues in CubeSats. In [76], for instance, the authors presented their experience with radiation testing on COTS components and the development of software to gain beneficial results during radiation testing. In addition, the authors emphasize the importance of understanding single-event effects on components and how FSW can be designed to tolerate them. In contrast, [77] and [78] presented the application of fuzz testing techniques to expedite the operational testing of CubeSats while maintaining their completeness. The authors also demonstrated the effectiveness of this approach by finding and solving bugs not covered by traditional strategies like unit testing and software in the loop simulation. While the work in [77] focused on three new 3U CubeSats under development at the University of Chile, the work in [78] discussed explicitly the application of fuzz testing techniques on the SUCHAI series of nanosatellites.

F. Communication and Networking

The three papers in this category [79–81] focus on different aspects of CubeSat and nanosatellite technology, emphasizing communication in the first place, followed by control and FSW. The first paper, [79], explored utilizing the Globalstar network for CubeSat and small satellite communications through the development of the LinkStar radio architecture and open-source flight management system QuickSAT/VMS. The second paper, [80], addressed the agile operation of prominent nanosatellite constellations with inter-satellite communications, proposing an evolutionary contact plan design using evolutionary algorithms to control the constellation operations. Lastly, [81] discussed the

FSW aspects of Microsats and Nanosats in the context of fast-paced, small-scale development environments. It highlighted the importance of satellite safe modes, configuration updates, on-orbit software upgrades, and security. While all three papers delve into the complexities of CubeSat and nanosatellite systems, the authors approach the subject matter from different perspectives. The first paper focuses on communication networks, the second on constellation control, and the other third on software-driven approaches to small satellite networks.

III. Review Analysis for FSW Design

To showcase the practical implementation of the best practices extracted from the literature of CubeSat FSW a proof of concept is presented as a case study, focusing on FSW Design and Development. The core of this investigation is to address the specific needs and challenges inherent in the design and implementation of CubeSat FSW.

A. Essential Features of an Effective FSW

This section identifies the desired features for the proposed FSW design and architecture by carefully examining existing works and solutions from the literature.

1. Modularity and Service-Oriented Architecture

A modular application, or app-based software architecture satisfies requirements, accelerates development, and allows future mission reuse by selecting needed apps. Commercial hardware companies offer modular FSW systems, but custom software interfaces may still be needed for scientific payloads [2]. Therefore, developing a custom FSW is typically inevitable. Yet, CubeSat missions, having different payloads, also have different FSW requirements. Thus, it is essential to have scalable and flexible software architectures where modules can be added, modified, or removed depending on the mission, without affecting the architecture [28]. In light of this, the Norwegian University of Science and Technology developed an FSW for a nanosatellite OBC [39]. The authors in [37] even asserted that modularity determines FSW quality. Consequently, they argued that all design aspects should be highly modular to enhance the flexibility and modifiability of the code.

Furthermore, many additional efforts were made to develop FSW for CubeSats with a heavy emphasis on modularity [3] [6] [27] [29] [34]. The authors in [4] summarized several published papers describing FSWs of CubeSat missions and added that “modularity, extensibility, flexibility, robustness, and fault-tolerance have been identified as the main features of the FSW for nanosatellites”.

2. Multi-layered Architecture

In addition to modularity, some works concluded that most of the FSWs were implemented as multi-layered architecture to exploit abstraction layers. For example, the FSW proposed in [37] is "divided into four modular layers: at the top are application-specific tasks, next are hardware specific libraries, at the bottom, are microcontroller specific drivers, and supporting all layers are general purpose software libraries." This approach is similar to the approach proposed in this work. However, the final FSW architecture differs from the proposed design due to dissimilarities in hardware. Reference [82] also demonstrated a multi-layered architecture.

3. Selection of Operating System

Simple embedded software has evolved into more sophisticated Embedded Operating Systems. Real-Time Operating Systems (RTOS) are emerging to address the challenges of multitasking, scheduling, and dynamic management of flight system resources [2]. The RTOS for CubeSat missions is usually chosen based on the microcontroller selected. Many of these Operating Systems are open-source libraries that can be *ported* to various microcontrollers and hardware platforms. Commonly used RTOS options include GNU/Linux or Free Real-Time Operating System (FreeRTOS). The choice of microcontroller and RTOS are interrelated, as a more powerful microcontroller is needed for FSW that requires high-level programming language or features only available in GNU/Linux, resulting in increased power consumption. On the other hand, using a lightweight RTOS can conserve power at the expense of some processing capabilities [4]. In the case study, the FSW is designed and implemented using the FreeRTOS [83], as in [3], [5], [31], [39], [42], and [55]. In addition, the work in [27] also uses FreeRTOS yet focuses heavily on the design of FSW robustness by employing Fault Detection, Isolation, and Recovery (FDIR) methods, aspects which are also investigated in this article.

4. Other Important Features

The works in [1] and [3] employ *bootloaders* for their FSWs for in-mission image update capability, as in the case study presented in this paper. However, their implementation of the bootloader is quite different in design. Furthermore, the work in [3] also implemented a Command Line Interface (CLI) used in their testing. The CLI is expanded upon in this paper, implementing it over the CubeSat Space Protocol (CSP) to form CLI/CSP, which is then used as the foundation for the script engine. CSP is a simple network and transport protocol stack written in C language. It was initially developed at Aalborg University in 2008 and is currently maintained by GomSpace. It is especially designed for satellite embedded systems [84].

B. FSW Guidelines, Frameworks, and Development Kits

The literature review concluded that many resources, including guidelines, frameworks, and FSW Development Kits (section A.1), are available for FSW development. However, while these tools allow expediting software development to some extent, there is not yet a widely adopted community standards for CubeSat FSW [2]. Nevertheless, before starting the implementation of the FSW, the following were reviewed in depth.

The comprehensive work in [31] attempted to develop a generic FSW and seemed promising. However, many generic and simulated works are published, such as [28], just to lay the foundation for future FSW developers without actually implementing and testing the FSW for actual satellite missions. Works as such remain theoretical and lack space heritage. The National Academies recognized this fundamental challenge in CubeSat FSW: numerous individuals attempting valuable work face difficulties in achieving space heritage. As a result, many developers are duplicating the effort rather than trust in existing solutions [2].

One of the few open-source and freely available frameworks is the well-established **NASA's cFS** [7]. In fact, this work's software design was inspired by NASA's cFS architecture. It employs a highly modular design where a software module and apps manage each functionality. Each app can be added or removed from the FSW based on the requirements, the mission, and the subsystems used. The work in [35] uses cFS to develop nanosatellite FSWs. However, it is technically challenging to configure and deploy cFS due to its rooted history with large and complex satellites [5]. The **OpenSatKit** was created to mitigate these problems [85]. OpenSatKit is a platform for developing and learning about NASA's cFS. However, it adds a learning curve with even additional complexity [5] [29]. Even though this suite is tested in NASA's nanosatellite missions, its adoption is still not broad enough. Many current CubeSat FSWs are still implemented directly (without cFS) over simpler lightweight operating systems such as FreeRTOS [28].

Another framework gaining popularity is **F Prime** developed by Jet Propulsion Laboratory (JPL) [8]. While cFS and F Prime offer substantial advantages for modularity and service-oriented architecture, they are not without their limitations in the context of CubeSat missions. cFS, for example, is deeply rooted in traditional, large-scale satellite systems, making it technically complex for CubeSat developers. Moreover, neither cFS nor F Prime inherently support FreeRTOS, a useful operating system for resource-constrained environments. As of now, cFS is primarily designed to run on RTOS like VxWorks, RTEMS, and Linux. This gap in compatibility limits flexibility for CubeSats with stricter hardware constraints. Additionally, neither framework offers a built-in bootloader or script engine, making it difficult

to update software during a mission or automate satellite operations. These gaps highlight the need for a more streamlined, CubeSat-specific framework like the one presented in this work, which directly addresses these challenges. Furthermore, the complexity in cFS and F Prime might increase development time for teams unfamiliar with these frameworks. The framework presented in this paper is designed to be more intuitive, helping lower the barriers for smaller teams or academic environments.

The FSW Development Kit (FSDK) by **Bright Ascension** [86] is also worth mentioning. This FSDK is a platform designed to expedite the creation of mission-specific FSW by leveraging a component-based architecture. This approach enables the reuse of pre-validated software components in various combinations. The FSDK was used in [87] to interface Clyde Space COTS components.

In partnership with the University of Patras, the **Libre Space Foundation** developed both the software and hardware for UPSat, a 2U CubeSat launched in 2017. Aligned with the foundation's main goal of offering open-source access to space technologies, the UPSat's FSW has been made publicly available [87]. The **LibreCube** [88] is an initiative designed to democratize space exploration by allowing everyone access to open-source hardware and software systems. Another open-source solution is **KubOS**, a platform that is a custom Linux distribution adapted to host the satellite applications [29]. Other attempts are promising yet still are work-in-progress, such as **CubedOS** [29], and **NanoSat MO** [89].

Moreover, the framework showcased in [33] uses **pysimCoder**, is an open-source tool used for developing real-time control applications. The main downsides of pysimCoder, when compared FSW architecture presented in this work, include its reliance on block diagram models for system development, which may limit the flexibility and depth of customization for complex CubeSat missions.

PyCubed [90] is an open-source hardware and software platform designed for CubeSat development, with a focus on simplicity. It provides a modular framework tailored for academic teams and small satellite developers, integrating features such as fault-tolerant power management and a streamlined software stack. By leveraging Python, PyCubed lowers the barrier to entry for CubeSat developers, making it a popular choice in educational and research environments. On the other hand, its reliance on CircuitPython and specific hardware configurations may limit performance, customization, and scalability for complex missions like PHI-Demo. The Python-centric design is less efficient for time-critical and resource-constrained tasks, and the platform's limited flight heritage raises concerns about reliability for high-stakes missions. Additionally, its community support and documentation are less extensive,

making it better suited for educational or experimental use rather than advanced CubeSat missions with demanding payloads and operational requirements.

Furthermore, the work described in [5] reviews other available FSW development frameworks and proposes a framework tailored explicitly for CubeSat missions. Similarly, reference [91] is a survey comparing six FSW frameworks based on 'New Space' criteria, aiming to identify the most suitable one for a nanosatellite mission.

IV. Case Study: 5G-Enabled, 12U CubeSat with Green Propulsion

Satellites are becoming a fundamental part of the Internet's infrastructure, especially the IoT. Therefore, the Mohammed Bin Rashid Space Center (MBRSC) has launched a Payload Hosting Initiative (PHI) program for Cube Satellites. The first iteration of this program is called PHI-Demo, a 12U CubeSat hosting a 5G-capable COM as a payload. As a secondary payload, it hosts a green propulsion technology to be tested in space. In the following sections, the design and architecture of FSW for PHI-Demo CubeSat is crafted and presented. This was achieved by first reviewing previous works in this realm to elicit the desired features for the FSW.

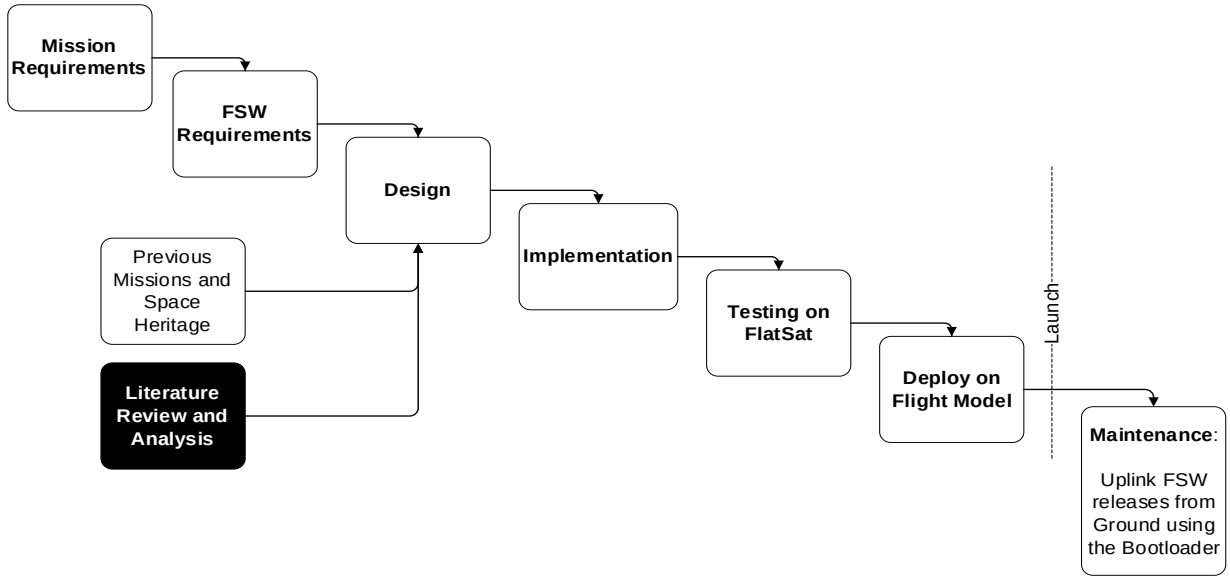

Fig. 5 A modified waterfall model depicting FSW development

The modified version of the waterfall model followed in this work for the FSW development is illustrated in Fig. 5. The process begins with defining the mission requirements which shape the FSW requirements. Next, influences from previous missions and the broader context of space heritage of MBRSC and the literature review and analysis presented in this work drove the design process. The subsequent stages of, implementation, and testing are executed

in a structured manner, each phase building on the preceding one. Notably, the testing on the FlatSat provides a critical simulation environment before deployment on the actual flight model. This step was thoroughly discussed and illustrated in the Testing Methodology section. The diagram culminates with the post-launch maintenance phase, highlighting the capability of uplinking FSW releases from the ground using the bootloader, thus demonstrating a closed-loop lifecycle from conception to operational management.

A. CubeSat Hardware Components

The CubeSat in the case study is a typical satellite that comprises various hardware components, including an OBC that runs the FSW, an EPS for energy collection and distribution, a COM for ground communication, an ADCS for location and orientation management, and payload subsystems designed to carry out 5G IoT communication, and propulsion.

1. The Onboard Computer

The "FSW typically runs on radiation-hardened processors and microcontrollers that are relatively slow and memory-limited" [10]. The CubeSat in this study uses an ARM Cortex M7 microcontroller from STMicroelectronics. This microcontroller out-of-the-box supports third-party software libraries such as FreeRTOS and FATFS, which are used in this work. Unfortunately, it has a RAM of only 1 MB, a limitation that must be kept in mind during the implementation. The ARM processor can be considered a System on Chip; The processor is combined with a set of peripherals on the die. These peripherals (such as UART, SPI, I^2C, and CAN) can be configured by writing to memory-mapped registers [1]. In addition, Error Correcting Codes (ECC) were enabled for the RAM and the Flash Memory, where the FSW images are stored.

2. The Subsystems

The subsystems that make up the CubeSat are as follows: an EPS that has a Distribution Board (DB) and two Power Conditioning and Control Modules (PCCM) for redundancy, A COM subsystem that has both UHV/VHF and S-Band capability, an ADCS Subsystem, two Payload Subsystems; one is a 5G IoT device, and the other is a green propulsion technology to be tested in space (PROP). Refer to Fig. 6 CubeSat Hardware Components.

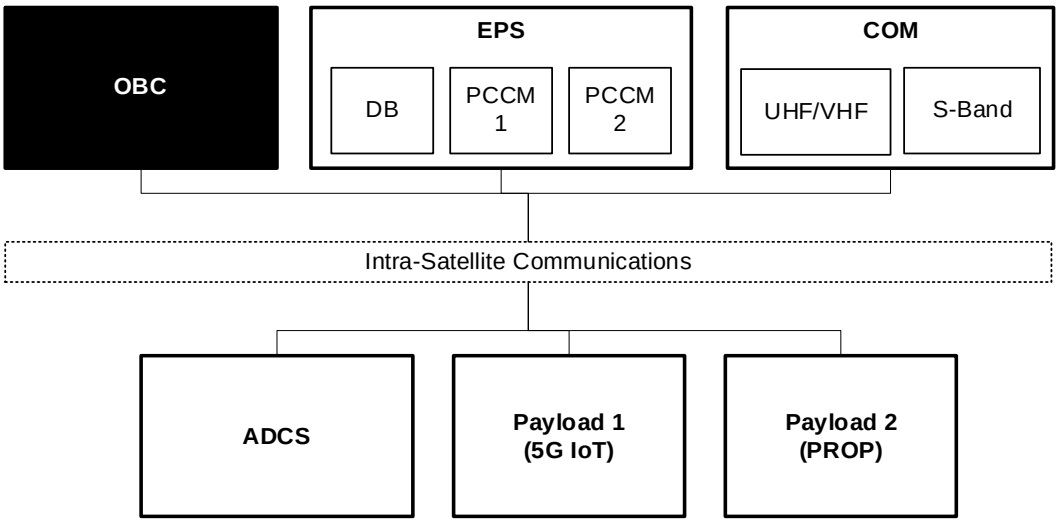

Fig. 6 CubeSat Hardware Components. Adapted from Eshaq, M., et al. [40]**, 2023 ASET, IEEE.**

Regrettably, CubeSats have a poor track record in terms of mission success. This is due to a common practice among researchers and developers, who tend to prioritize subsystems and hardware design over the development of FSW and the integration of the satellite as a complete system. To address this issue, it is crucial to invest more resources into the architecture, design, and implementation of FSW, as well as the development of fault tolerance mechanisms [3].

V. FSW Design and Architecture

The FSW comprises various Applications (Apps) and supporting libraries, each with a specific purpose and function. The aim is to create a modular and service-oriented architecture where Apps can be added or removed based on the mission requirements and reused for future missions. The FSW architecture is illustrated in **Fig.** 7 below.

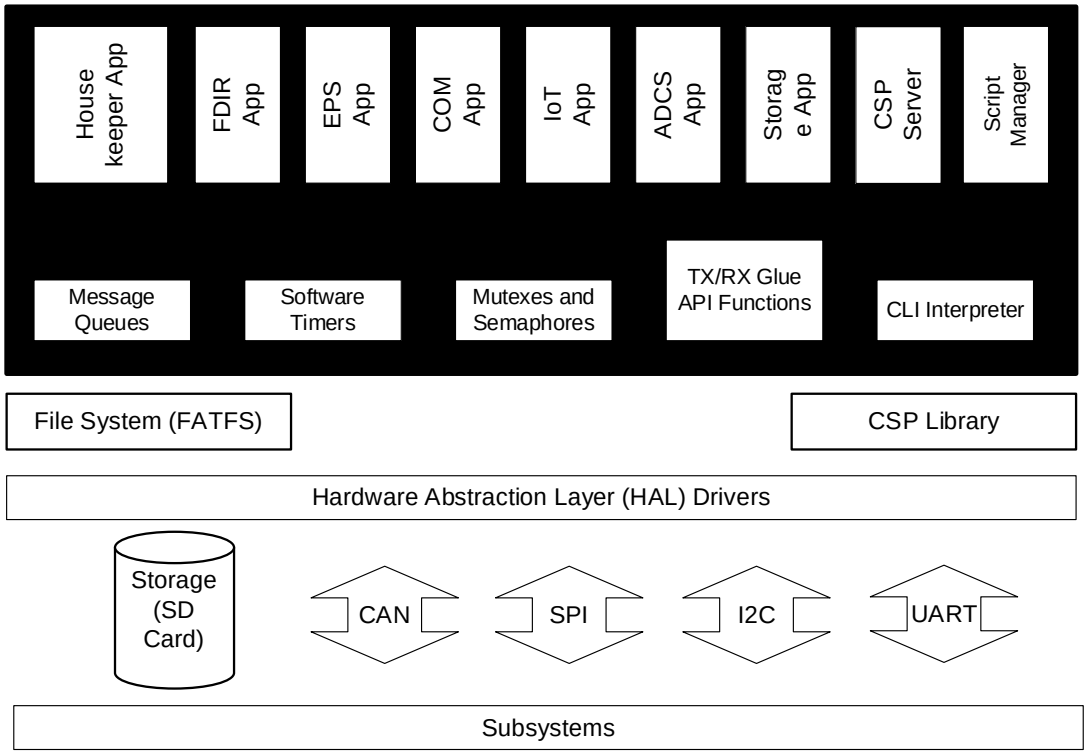

Fig. 7 Layered Software Model. Adapted from Eshaq, M., et al. [40]**, 2023 ASET, IEEE.**

The **Housekeeper App** collects spacecraft telemetry and stores it as Whole Orbit Data (WOD) files. This app is critical for ensuring the overall health and performance of the spacecraft, making it an essential component of the FSW in a CubeSat mission. The **FDIR App** continuously monitors the telemetry and determines the spacecraft's health. The FDIR employs Watch Points (WP) and Action Points (AP). The WP are subsets of telemetry that are constantly monitored to ensure they remain within safe levels for the spacecraft's nominal operation. Each WP has an AP linked to it. If a WP falls outside the safe range, the corresponding AP is executed to recover from the detected fault. If recovery fails, the spacecraft enters a 'safe mode' until ground intervention.

The **EPS App** manages the solar panels, batteries, and the power distribution. The **COM App** manages all satellite communication activities. The **Payload (PLD) App** manages the payloads. It provides an interface to the payload and handles its data flow. The **ADCS App** governs the spacecraft's attitude by controlling the ADC subsystem. Finally, the **Storage App** manages and ensures the integrity of the storage unit (SD Card). It mounts and initializes a File Allocation Table 32 (FAT32) volume, handles all commands related to files and systems from the ground and carries out file transfers (large data transfers) via Uplink and Downlink.

A **CSP Server App** is critical to the FSW. It accepts all incoming CSP connections to the OBC, whether the connection is coming from an onboard subsystem or ultimately from the ground via the COM subsystems. Several dedicated CSP ports are assigned and used. One port is dedicated to receiving and processing CLI commands sent in CSP packets, explained in the 'CLI Commands over CSP' section below. Another very central component of the satellite operation is the **Script Manager App**. Details on this App are highlighted in section D.

```
Algorithm CubeSat Flight Software Operation
Input: Libraries, Hardware Initializations, FreeRTOS, CSP
Output: Running CubeSat Operations

1. Initialize the system including hardware, FreeRTOS, and CSP
2. Start the FreeRTOS kernel leading to parallel execution of applications
3. Define Applications (Apps) for specific tasks
   - Housekeeper App:
     a. Collect telemetry from all subsystems periodically
     b. Store collected telemetry into Whole Orbit Data (WOD) file
   - FDIR App:
     a. Monitor Watch Points (selected telemetry) against predefined triggers
        and thresholds
     b. Execute corresponding Action Points as required, potentially putting
        the spacecraft in Safe Mode
   - CSP Server App:
     a. Await and process incoming packets from Ground
     b. Distribute packets or commands to respective Apps via message queues
        if not addressed to the server
   - Other Apps:
     a. Check for and process messages in the message queue
     b. Perform app-specific operations or remain in idle state
4. Repeat the process for each App, ensuring continuous operation
5. Utilize sleep or delay as necessary to manage operation timing and resource usage
```

Fig. 8 FSW Apps Pseudocode

The pseudocode in Fig. 8 highlights the code skeleton for selected apps such as, the Housekeeper, FDIR, and CSP Server App, while also providing a generic pseudocode at the end for the other apps which roughly follow the same structure. The script engine was excluded from this figure since its operation is thoroughly illustrated later in the Script Engine section.

As for the choice of programming language for this work, using C for a CubeSat mission offers several advantages: it's widely supported, efficient for low-power and limited-resource environments like those found in CubeSats, and provides direct control over hardware for performance-critical applications. Its extensive libraries and broad community support facilitate solving complex problems and interfacing with various hardware components. The use

of C language for flight software development has been demonstrated, particularly in the context of CubeSat missions [3] [4]. Moreover, it is worth noting that NASA's cFS, CSP library, and FreeRTOS are all C-based. In light of this, the FSW in this study was written in C language.

A. Operating System

The operating system of choice was the FreeRTOS, distributed freely under the MIT open-source license [83]. FreeRTOS was selected due to many reasons. One was that it has recently gained popularity, especially in CubeSat missions, as established in section 3. Second, FreeRTOS was also chosen because the microcontroller natively supports it. Finally, FreeRTOS is also often chosen due to its simplicity and large user community in the field of satellites and other embedded systems [39].

B. File System

The File Allocation Table File System (FATFS) is a basic file system module for embedded systems that is also Windows-compatible [83]. It is a software library that allows managing and organizing files on a storage device, such as a disk drive or memories. In this project, the FAT32 variant of the file system is used due to its ability to handle larger storage sizes and a more significant number of files. The low-level disk I/O modules are entirely separate from the FATFS module [92]. FATFS is a middle layer that allows the FSW to be written independently of the underlying media storage device drivers or host controllers. The adoption of this middle layer aligns with the intended Multi-layered Architecture. FATFS was selected for this project due to its simplicity and native support by the Microcontroller and IDE. In addition, having a file system allows all Apps to have the ability to record events to their designated log files.

C. Command Line Interface

To simplify commanding the satellite, a CLI layer is added, which provides a layer of abstraction that allows planners and operators on the ground to control the spacecraft without having to work with the many native binary commands and responses used in each subsystem. The CLI commands introduced are high-level and human-readable, implemented with the help of the FreeRTOS-Plus CLI Framework. Examples of such commands are presented in Table 2.

Table 2. CLI Commands and Replies

CLI Command (Input)	Expected Reply (Output)	Description
`obc get time`	`OK 1632133618`	Get time in seconds using Unix epoch time format.
`obc set time 1632133620`	`OK`	Set current OBC time in seconds using Unix epoch time format.
`adcs point-to nadir`	`OK`	Maneuver the Sat to Nadir.
`eps iot on`	`OK`	Power ON the IoT Payload.
`delay 15 mins`	`OK`	Delay the Script Engine for 15 mins.
`eps iot off`	`OK`	Power OFF the IoT Payload.
`msat set mode safe`	`OK`	Change the Sat mode to safe.
`boot install FSW.bin 2`	`Installation COMPLETE`	Install the FSW image to slot 2.
`boot from image 2`	`OK`	Boot from the FSW in slot 2.
`script run sequence.scr 1`	`OK`	Run script stored in the file starting from line 1.

1. Command Console

Once CLI commands are defined in the FreeRTOS *CLI Interpreter*, they can be invoked anywhere within the FreeRTOS environment. Apps (FreeRTOS tasks) can execute commands by sending a 'string' containing the command and any required parameters to the CLI interpreter function. The command is then processed and executed, and a reply is returned to the caller. In light of this, a Command Console was implemented to exploit this mechanism. When testing the satellite on the ground (in a FlatSat test environment), the OBC was connected to via a serial port and a terminal. Then, the commands were typed in and executed, and outputs could be observed in real-time, as shown in Fig. 9. This allowed the team to test all of the CLI commands defined and registered and observe the output, without the need for a complete Mission Control Software (MCS) to be implemented and ready for testing. The Command Console also assisted in the Assembly, Integration, and Testing (AIT) process and in testing the operation scenarios of the satellite as a whole.

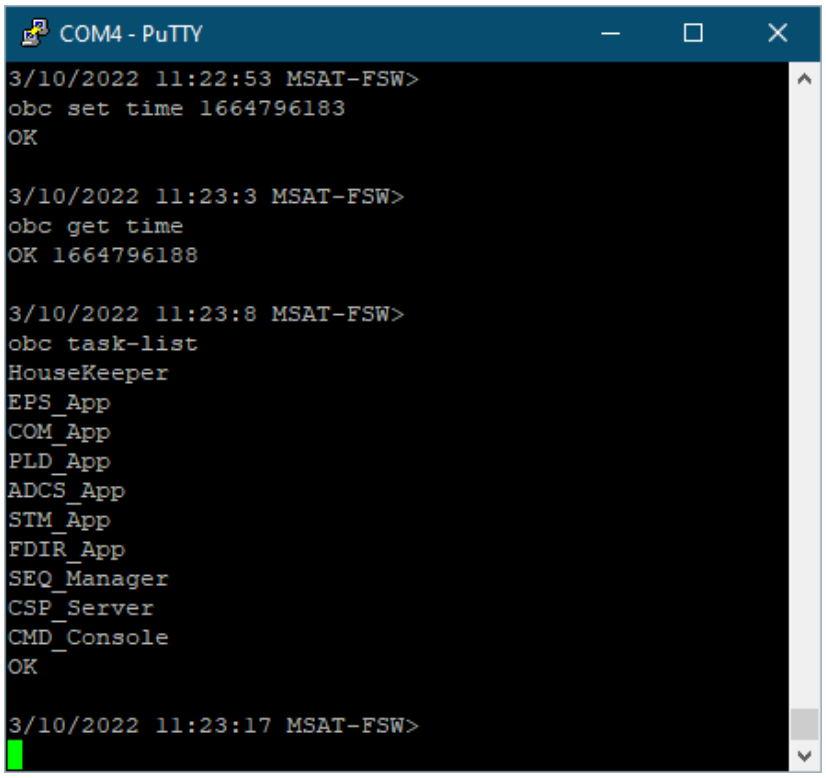

Fig. 9 Command Console via Serial Terminal. Reprinted from Eshaq, M., et al. [40]**, 2023 ASET, IEEE.**

2. CLI Commands over CSP

The CSP server is designed to receive CSP packets that also contain CLI command strings. Each command is executed upon arrival, and a response is subsequently transmitted back to the ground station within a separate CSP packet. This allows the ground team, using the MCS, to interact with the satellite during contact periods with CLI commands. This interaction closely mirrors the experience of employing the Command Console, as previously described.

The command line abstraction layer effectively conceals the satellite's underlying hardware specifications and communication protocols. As a result, the MCS no longer requires knowledge of the specific hardware architecture or the intra-satellite communication protocols between the OBC and subsystems. This facilitates the potential for the MCS to be repurposed for future missions, enhancing its reusability.

D. Script Engine

A script file can be uploaded to the satellite pre-launch or upon contact. A script file is simply a sequence of CLI commands optionally separated by time delays and stored in a text file. When a script file is triggered, commands are released and executed in a timely fashion while orbiting, whether the satellite is still in contact or not.

The script engine works by first setting the engine itself to *armed*. A script file must also be marked as *armed*. This means that the file is now ready to be executed. Arming must be done by ground or by a previously running script. Next, the script engine reads the armed file and starts executing the (input) CLI commands stored in the script file line-by-line. Upon execution, an output file is created and filled with (output) reply lines corresponding to each CLI command executed. Upon contact, the ground team can then download and examine the output file, and new script files for subsequent satellite operation can be uploaded. At the end of each script file, another script file can also be ‘armed’. Upon finishing the execution of the first one, the following script file is executed. This allows the ground team to queue or *chain* many script files covering satellite operations for many orbits spanning many upcoming days or weeks. This feature means satellite operations do not have to be hardcoded in the FSW. Instead, satellite operations can be defined as scripts and potentially reused in future missions. This mechanism will govern the operation of the CubeSat while orbiting in the mission as well as the operation and communication of the 5G IoT payload. Propulsion firing can also be managed by a script file. Operators can upload scripts during contacts to start the payload operation at predetermined time schedules.

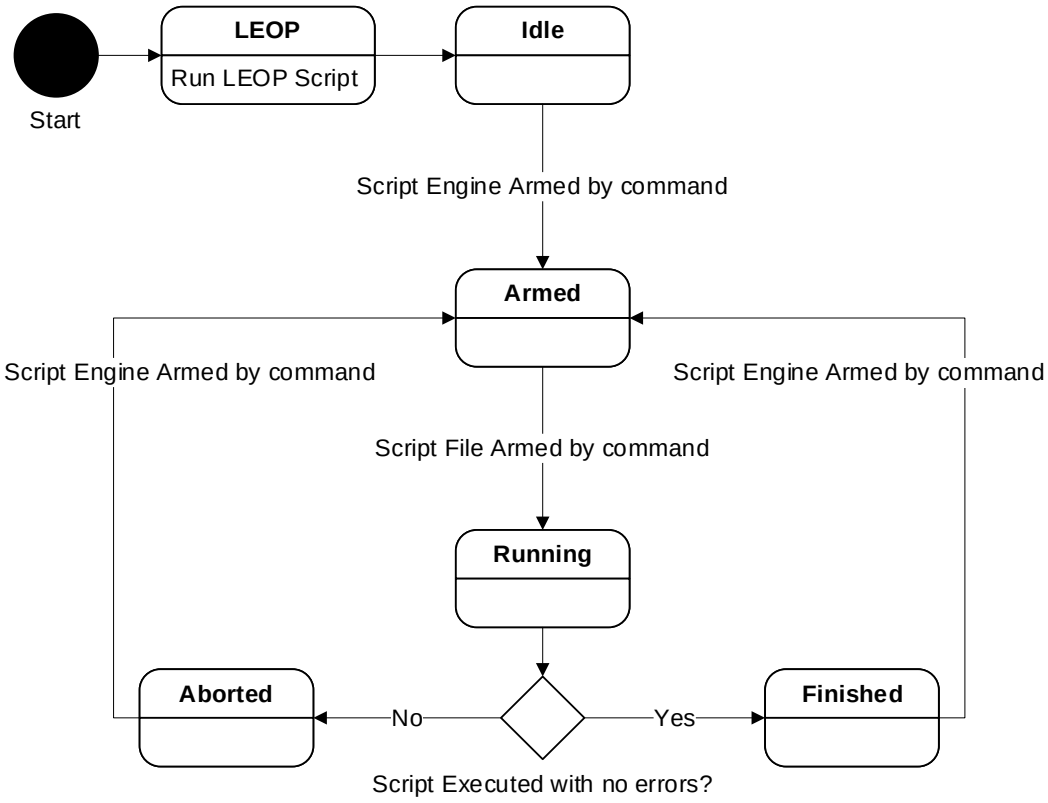

Fig. 10 State Machine Diagram for the Script Engine

Similarly, when the FSW is initialized and run for the first time (upon separation), a script dedicated for Launch and Early Orbit Phase (LEOP) operations is run immediately. The LEOP script conducts operation such as solar panel and antenna deployment. Then, the script engine enters an idle mode waiting to be armed, as shown in Fig. 10.

E. Bootloader

A bootloader is a small piece of software that runs before the main FSW and is responsible for checking and executing the FSW. With a bootloader, a CubeSat can be updated with new FSW while in orbit. The FSW running on the OBC must be highly reliable, as maintenance is almost impossible after launch. Scenarios leading to faults and errors, and ultimately mission failure, must be predicted beforehand, and solutions to such issues should be prepared for in advance [39]. A bootloader can provide a safety net if the FSW becomes corrupted or fails. The bootloader can detect a failure to load an FSW image and load a backup version, providing a fail-safe mechanism for the spacecraft. This makes the ability to modify/update FSW while in-mission a highly desirable feature. It may not directly contribute

to the modularity of the FSW; however, it certainly adds flexibility to its design and development process and makes the FSW future-proof and thus reusable.

The bootloader has four slots for storing FSW images. The first slot contains the main image (Golden Image), installed pre-launch, and can never be overwritten. Other slots may store additional FSW images as a form of software redundancy and backup. If an image is corrupt, the bootloader can boot from other images. Refer to Fig. 11.

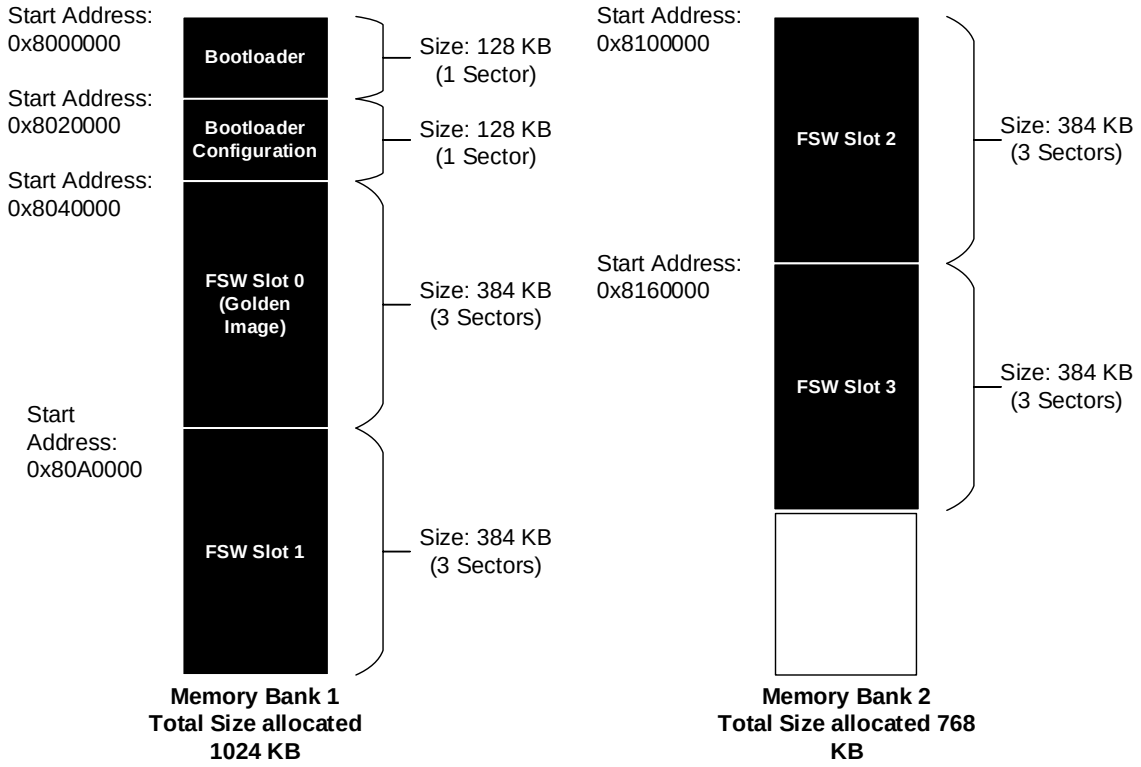

Fig. 11 Memory Organization for Bootloader in Flash Memory

New images can be uploaded to the spacecraft as a binary file during a mission. After uploading, the image must then be installed in any of the other slots. The spacecraft is then instructed to boot from the newly uploaded image. The bootloader then starts by attempting to boot from the newly uplinked image. If it fails, it attempts to boot from the image the satellite managed to boot from the last time. If that fails, it will look for images in other slots. In each attempt, the bootloader checks the slot area in the memory and sees if the slot is empty or not, then makes three attempts to boot from it. Finally, if all of the above fails, the bootloader defaults to the Golden Image and boots from it (as illustrated by Fig. 12). The bootloader achieves this by employing a watchdog. A watchdog is a hardware timer that monitors the operation of the system. When a fault or an unknown state occurs, the watchdog timer will time out

(after 4 seconds, for instance), and it will restart the system or put the system back into a known state from which the system can recover. Watchdogs have been used in terrestrial and space systems and can improve system reliability in CubeSats [93]. When the bootloader starts, the watchdog timer is also started, and a crash counter is incremented. Successful execution of the FSW will clear the watchdog timer and reset the crash counter. If the watchdog timer is not cleared within a given time window, the watchdog will reset the OBC, forcing it to enter the bootloader again to decide what to do next. A crash counter higher than a certain threshold (3 counts, for instance) indicates that the image is unstable, and the bootloader should choose another image instead.

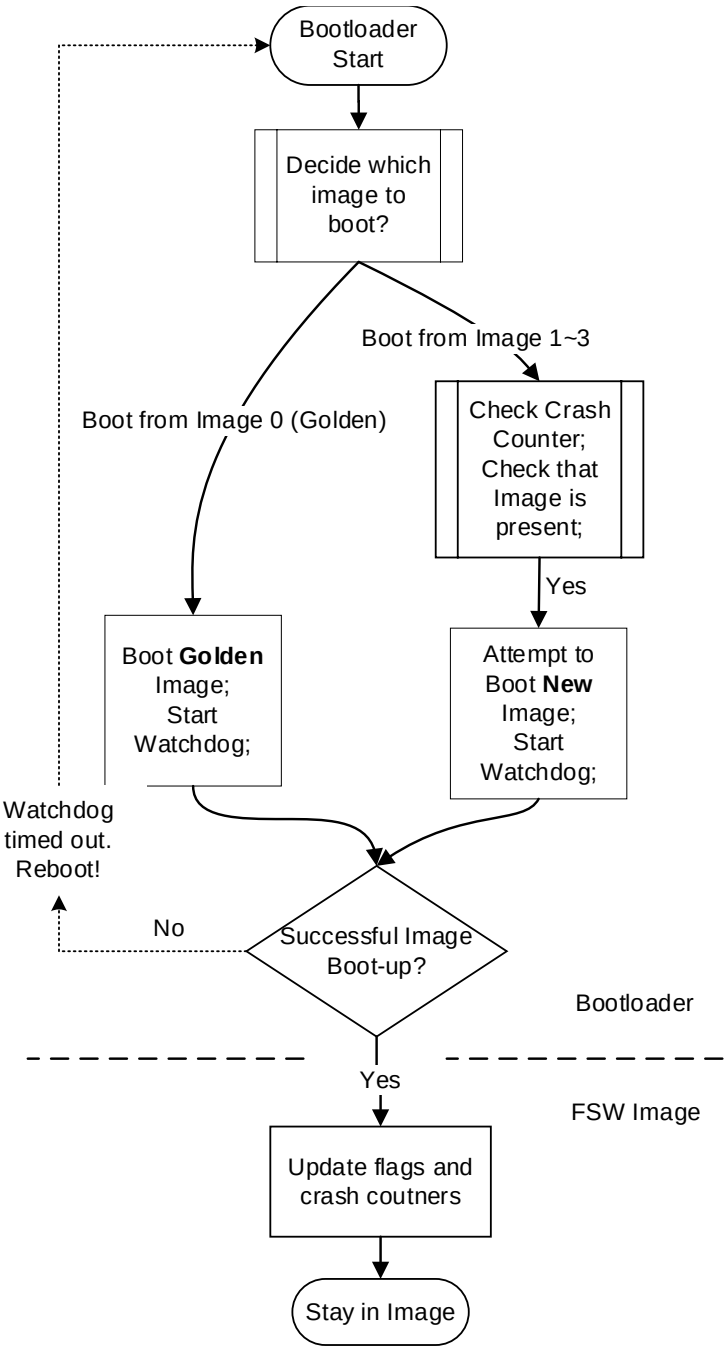

Fig. 12 Bootloader Flowchart

VI. Testing and Results

A substantial portion of project time should be dedicated to FSW Testing. Identifying and rectifying defects during the initial implementation phase offers substantial benefits, yielding savings of no less than ten times in contrast to detecting them during integration or, even less favorably, post-launch [39]. Furthermore, the challenging space

environment presents a multitude of potential system failures due to ionized particles that have the capacity to trigger catastrophic software crashes and power outages. Additionally, the considerable fluctuations in temperature can lead to the degradation of batteries and other crucial components [28]. It's worth noting that CubeSats have historically demonstrated a less than optimal mission success rate [3], thus necessitating significant endeavors to enhance their robustness [28].

A. Test Setup

The Eclipse-based STM32CubeIDE was used for code development. STM32CubeIDE uses the GCC toolchain for development and GDB for debugging [94]. Furthermore, FreeRTOS version 10.3.1 was used, with CMSIS version 2.00. The RTOS was set to Cooperative Scheduling. The memory scheme used was heap 4. As a rule of thumb, using dynamic memory in embedded systems is not recommended. Nonetheless, a study focused on assessing various memory schemes within the context of the CubeSat project reached the determination that heap 4 presented the most optimal suitability. This selection was based on its ability to strike a favorable equilibrium between code size overhead and segmentation level considerations [39].

B. Testing Methodology

The objective was to ensure that the FSW meets all of the functional requirement and the non-functional requirements. Test cases for the mission FSW were driven by analyzing both types of requirements. These test cases were designed to thoroughly evaluate the performance and capabilities of the FSW in various mission scenarios. These test cases were executed and meticulously documented the results to assess the FSW’s conformance to its requirements and identify any potential issues or improvements needed. In addition to the individual test cases, a long-term test was conducted for the FSW, simulating a prolonged mission environment using a FlatSat setup. During this extensive test, which lasted over ten days, the software operated continuously, emulating the conditions it would encounter during actual mission operations. Throughout this period, all software activities and events were logged, providing valuable insights into the FSW’s long-term stability, reliability, and performance. This comprehensive testing approach allowed us to identify and address any potential issues, ensuring that the FSW is well-prepared for the challenges it will face during the actual CubeSat mission. **Fig.** 13 highlights the testing process and methodology.

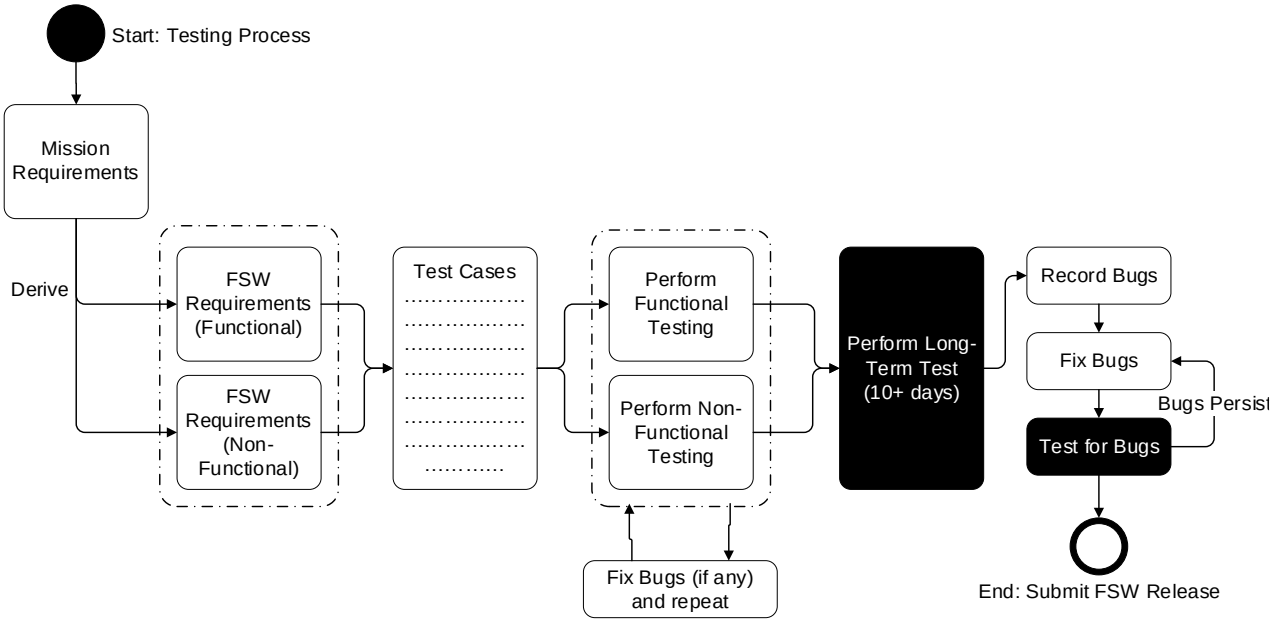

Fig. 13 FlatSat Testing Methodology

During the testing process, no instances of crashes were observed in which the FSW became entirely unresponsive and incapable of recovering on its own. Three distinct types of bugs were identified that occurred during the test runs; however, the FSW was able to independently recover from each of these issues, demonstrating its resilience and fault tolerance capabilities. In one particular instance, a memory leak event was detected. Following this observation, measures to address the identified bugs were taken, implementing appropriate fixes to prevent similar issues from arising in the future. Moreover, an FDIR Watch Point/Action Point (WP/AP) was introduced to the FSW, ensuring that it could promptly rectify the situation if a similar event were to occur during actual mission operations. This proactive approach to addressing potential problems enhances the overall reliability and robustness of the FSW, contributing to mission success and longevity.

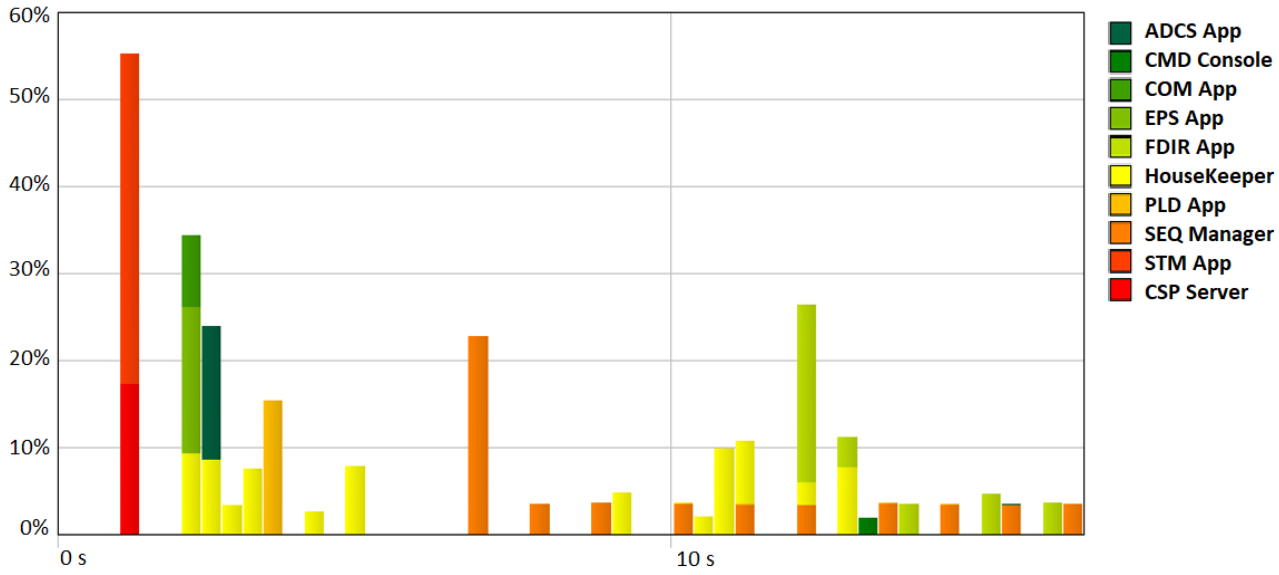

Fig. 14 CPU Utilization

The graph depicted in **Fig.** 14 illustrates the CPU usage over time for each individual task during the initial 16 seconds of FSW startup. A noteworthy observation is that, even with the inclusion of all features in the FSW, the CPU utilization on average remained below the 40% threshold. This finding indicates that the FSW effectively manages system resources, ensuring efficient performance without overwhelming the CPU, which is vital for maintaining the overall stability and reliability of the CubeSat system during its mission.

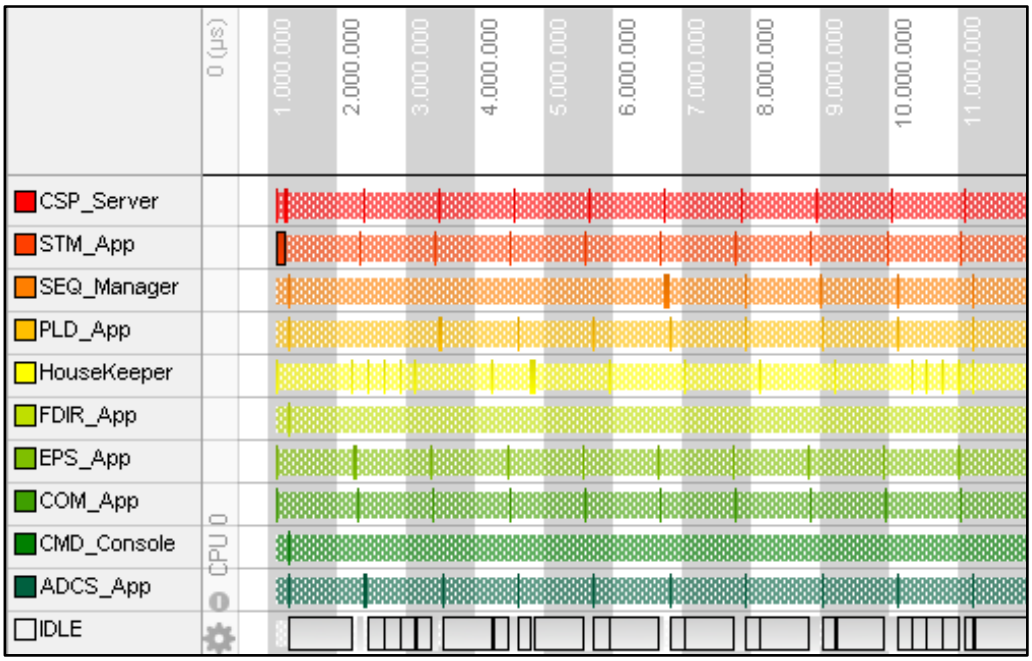

Fig. 15 Trace View of CubeSat FSW Task Execution with One-Second Intervals. Reprinted from Eshaq, M., et al. [40]**, 2023 ASET, IEEE.**

Furthermore, Fig. 15 presents a trace view, which is essentially a timeline visualization of how various tasks within the flight software are being executed over time. It's interesting to see how tasks are scheduled and managed, with clear indications of their execution periods and any overlaps that may occur. This type of trace is crucial for ensuring that tasks are completed within their allotted time slots and that the system meets the real-time requirements of the mission. The periodic delay of one second across tasks aligns with constraints from prior missions and suggests a deliberate pacing to accommodate processing or communication needs.

C. Discussions

The aim was to ensure the robustness of the software while meeting the FSW requirements. The most crucial phase of our testing was the long-term test lasting over 10 days. The idea was to keep the satellite as a whole (in a FlatSat setup) running while performing all the tasks that it is expected to perform in the mission and observe the behavior. The FSW recovered from faults as expected. The test engineers were allowed to interact with the satellite pretending that they were in contact. No physical interactions (such as power cycling) were allowed with the FlatSat.

The subsequent tests and results gathered were to ensure that the OBC was not overloaded and running the FSW with ease. This is why CPU utilization was monitored as well as memory consumption. Faults accumulate and result

in failures when systems run at their maximum potential. Moreover, the intention of this work was to demonstrate that FSW for CubeSats can be packed with features (such as script engines) without compromising performance. Modern CubeSat CPU and hardware allows for this, and FreeRTOS is lightweight indeed.

The testing methods offered valuable insights and validation, yet they also had some limitations. First, the long-term testing period, although extensive, spanned just over 10 days. This duration may not fully capture potential issues that could arise over a longer, more realistic mission period. Additionally, the testing environment, using a FlatSat setup, though designed to simulate the conditions of a space mission, cannot replicate the challenges and complexities of the actual space environment. This includes factors like radiation effects, thermal variations, and unexpected hardware anomalies.

In the realm of CubeSat development, while mechanical and electronic interfacing of components is guided by established standards, the flight software (FSW) lacks a clear and standardized development roadmap [2]. This situation is compounded by the tendency within the engineering community to allocate insufficient focus and attention to FSW and its critical testing phase [78]. This oversight contributes to the common occurrence of software failures in CubeSat missions, highlighting a critical area for improvement in the pursuit of reliable and efficient space exploration endeavors.

VII. Conclusion

In conclusion, this study highlights the growing interest and continuous advancements in CubeSat flight software (FSW) development. A comprehensive analysis of the literature reveals a steady rise in publications, demonstrating a strong commitment to exploring various facets of CubeSat technology and its associated FSW. Key areas of investigation encompass design and development methodologies, subsystems, component interfacing, reliability, fault tolerance, anomaly analysis, testing, validation, communication, and networking. It is evident that the development of robust and efficient FSW is fundamental to the success of CubeSat missions. However, despite the progress made in this field, a significant challenge remains—the absence of freely-available, truly modular, and reusable FSW solutions and frameworks. This gap necessitates students and enthusiasts to undertake FSW development, often from scratch, impeding innovation and efficiency. Addressing this challenge requires the adoption of characteristics such as service-oriented architecture layered software, such as the FSW presented in this work. Additionally, standardizing communication protocols, particularly the use of the CubeSat Space Protocol (CSP), can significantly contribute to FSW modularity. Looking ahead, the future of CubeSat FSW development is envisioned to be greatly streamlined

through modular, portable, and reusable solutions. The incorporation of features like Command Line Interfaces (CLIs), Script Engines, and Bootloaders will empower future space engineers and researchers to build efficient and adaptable FSW with reduced development time. Moreover, the use of CSP as a standard for intra-satellite and ground communication is poised to become a cornerstone of CubeSat FSW design. In sum, this study not only sheds light on the current state of CubeSat FSW but also points to a promising future where modular, reusable solutions will foster innovation and excellence in small satellite technology.

References

[1] Kekez, D. D., "Development of Flight Software and Communication Systems for the CanX-2 Nanosatellite," University of Toronto, Toronto, 2006.

[2] National Academies of Sciences, E. and M., "Achieving Science with CubeSats: Thinking Inside the Box," *Achieving Science with CubeSats*, 2016. https://doi.org/10.17226/23503

[3] de Souza, K. V. C. K., Bouslimani, Y., and Ghribi, M., "Flight Software Development for a CubeSat Application," *IEEE Journal on Miniaturization for Air and Space Systems*, Vol. 3, No. 4, 2022, pp. 184–196. https://doi.org/10.1109/jmass.2022.3206713

[4] Gonzalez, C. E., Rojas, C. J., Bergel, A., and Diaz, M. A., "An Architecture-Tracking Approach to Evaluate a Modular and Extensible Flight Software for CubeSat Nanosatellites," *IEEE Access*, Vol. 7, 2019, pp. 126409–126429. https://doi.org/10.1109/ACCESS.2019.2927931

[5] El Allam, A. K., Jallad, A. H. M., Awad, M., Takruri, M., and Marpu, P. R., "A Highly Modular Software Framework for Reducing Software Development Time of Nanosatellites," *IEEE Access*, Vol. 9, 2021, pp. 107791–107803. https://doi.org/10.1109/ACCESS.2021.3097537

[6] Hansen, L. J., and Hanson, J., "Reusable, Modular, and Scalable Flight Software," *IEEE Aerospace Conference*, 2014. https://doi.org/10.1109/AERO.2014.6836259

[7] Lisa Kane, "Core Flight System," NASA, 2023. Retrieved 4 April 2023. https://cfs.gsfc.nasa.gov/

[8] Bocchino, R. L., Canham, T., Watney, G. J., Reder, L. J., and Levison, J., "FPrime: An Open-Source Framework for Small-Scale Flight Software Systems," *32nd Annual AIAA/USU Conference on Small Satellites*, 2018. Retrieved 6 April 2023. https://core.ac.uk/download/pdf/220136003.pdf

[9] Michael Swartwout, "CubeSat Database," 2023. Retrieved 7 May 2023. https://sites.google.com/a/slu.edu/swartwout/cubesat-database

[10] Dvorak, D. L., "NASA Study on Flight Software Complexity," 2009.

[11] Doyle, M., Gloster, A., Griffin, M., Hibbett, M., Kyle, J., O'Toole, C., Mangan, J., Murphy, D., Wong, N. L., Akarapu, S. K. R., Dunwoody, R., Erkal, J., Finneran, G., Reilly, J., Salmon, L., Thompson, J., Walsh, S., Shortt, B., Martin-Carrillo, A., McBreen, S., McKeown, D., O'Connor, W., Uliyanov, A., Wall, R., and Hanlon, L., "Design, Development and Testing of Flight Software for EIRSAT-1: A University-Class CubeSat Enabling Astronomical Research," *Journal of Astronomical Telescopes, Instruments, and Systems*, Vol. 9, No. 1, 2023, p. 017002. https://doi.org/10.1117/1.JATIS.9.1.017002

[12] Venturini, C., Braun, B. M., Hinkley, D., and Berg, G., "Improving Mission Success of CubeSats," June 2017.

[13] Dániel, V., Svoboda, P., Junas, M., Sabol, M., Cagáň, J., Stejskal, M., Dudas, J., Mikulickova, L., Marek, A., Pavlica, R., Sedláčková, P., and Jech, D., "Development of CubeSat with COTS Camera Enabling EO with High GSD," *Sensors, Systems, and Next-Generation Satellites XXIV*, Vol. 11530, 2020, pp. 179–187. https://doi.org/10.1117/12.2575862

[14] Nagarajan, C., D'Souza, R. G., Karumuri, S., and Kinger, K., "Design of a Cubesat Computer Architecture Using COTS Hardware for Terrestrial Thermal Imaging," *2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology*, 2014, pp. 67–76. https://doi.org/10.1109/ICARES.2014.7024379

[15] Mireault-Lecourt, C., Pelletier, E. C., and Laurin, J. J., "Study of a SAR Mission Onboard a 12U CubeSat Using the Reflectarray Technology," *2021 IEEE 19th International Symposium on Antenna Technology and Applied Electromagnetics, ANTEM*, 2021. https://doi.org/10.1109/ANTEM51107.2021.9518994

[16] Bauer, A., Viard, T., Liu, Y., and Rolland, J. P., "Freeform Hyperspectral Imager Design in a CubeSat Format," *Optics Express*, Vol. 29, No. 22, 2021, pp. 35915–35928. https://doi.org/10.1364/OE.439530

[17] Munoz-Martin, J. F., Fernandez, L., Ruiz-De-Azua, J., and Camps, A., "The Flexible Microwave Payload -2: Architecture and Testing of a Combined GNSS-R and L-Band Radiometer with RFI Mitigation Payload for Cubesat-Based Earth Observation Missions," *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2019, pp. 5185–5188. https://doi.org/10.1109/IGARSS.2019.8898402

[18] Burghardt, T., Picquet, R., Ternus, K., Shah, U., Mejias, H., Rodriguez, D., Ketchersid, S., Zarandi, K., and Juganu, N., "Space-Weather Monitoring, 12U CubeSat Design," *AIAA Southeastern Regional Student Conference*, 2021. Retrieved 12 May 2023. https://commons.erau.edu/cgi/viewcontent.cgi?article=1110&context=aiaar2sc#:~:text=The%20proposed%2012U%20CubeSat%20concept,Coronal%20Mass%20Ejections%20(CMEs).

[19] Massaro Tieze, S., Liddell, L. C., Santa Maria, S. R., and Bhattacharya, S., "BioSentinel: A Biological CubeSat for Deep Space Exploration," *Astrobiology*, 2020. https://doi.org/10.1089/AST.2019.2068

[20] "GOMspace | Software Defined Radio (MK2)," 2023. Retrieved 18 May 2023. https://gomspace.com/shop/phased-out-products/software-defined-radio.aspx

[21] "Versatile Wideband Software Defined Radio (SDR) | EnduroSat," 2023. Retrieved 18 May 2023. https://www.endurosat.com/cubesat-store/cubesat-communication-modules/versatile-wideband-sdr/

[22] Medina, I., Santiago, L., Hernández Gómez, J. J., Castillo, R., and Couder Castañeda, C., "Artificial Vision Algorithm for Earth Station Location from CubeSats in Narrow Beam Optical Communications," *Journal of Physics: Conference Series*, Vol. 2307, No. 1, 2022, p. 012023. https://doi.org/10.1088/1742-6596/2307/1/012023

[23] Hassan, N. B., Ali, M., Griffiths, A. D., Lowe, C., MacDonald, M., Dawson, M. D., Herrnsdorf, J., and Strain, M. J., "Integration of an LED/SPAD Optical Wireless Transceiver with CubeSat On-Board Systems," *Proceedings of the 2020 IEEE Photonics Conference, IPC 2020*, 2020. https://doi.org/10.1109/IPC47351.2020.9252367

[24] Griffiths, A. D., Herrnsdorf, J., Lowe, C., Macdonald, M., Henderson, R., Strain, M. J., and Dawson, M. D., "Temporal Encoding to Reject Background Signals in a Low Complexity, Photon Counting Communication Link," *Materials 2018*, Vol. 11, No. 9, 2018, p. 1671. https://doi.org/10.3390/MA11091671

[25] Pascoa, J. C., Teixeira, O., and Filipe, G., "A Review of Propulsion Systems for CubeSats," *ASME International Mechanical Engineering Congress and Exposition, Proceedings (IMECE)*, Vol. 1, 2019. https://doi.org/10.1115/IMECE2018-88174

[26] Adams, C., Parker, J., and Cotten, D., "A Hardware Accelerated Computer Vision Library for 3D Reconstruction Onboard Small Satellites," *IEEE Aerospace Conference*, 2021. https://doi.org/10.1109/AERO50100.2021.9438159

[27] Latachi, I., Rachidi, T., Karim, M., and Hanafi, A., "Reusable and Reliable Flight-Control Software for a Fail-Safe and Cost-Efficient Cubesat Mission: Design and Implementation," *Aerospace*, Vol. 7, No. 10, 2020, p. 146. https://doi.org/10.3390/AEROSPACE7100146

[28] Araguz, C., Marí, M., Bou-Balust, E., Alarcon, E., and Selva, D., "Design Guidelines for General-Purpose Payload-Oriented Nanosatellite Software Architectures," *Journal of Aerospace Information Systems*, Vol. 15, No. 3, 2018, pp. 107–119. https://doi.org/10.2514/1.I010537

[29] Quiros-Jimenez, O. D., and D'Hemecourt, D., "Development of a Flight Software Framework for Student CubeSat Missions," *Revista Tecnología en Marcha*, 2019. https://doi.org/10.18845/tm.v32i8.4992

[30] Mahmood, R., Khurshid, K., and Ul Isalm, Q., "Design Techniques for On-Board Software of Nano-Satellites," *2013 6th International Conference on Recent Advances in Space Technologies (RAST)*, 2013, pp. 517–522. https://doi.org/10.1109/RAST.2013.6581263

[31] Heunis, A. E., "Design and Implementation of Generic Flight Software for a CubeSat," Stellenbosch University, 2014.

[32] Marquez, S., Asundi, S., and Chiu, K., "Model-Based CubeSat Flight-Software Architecture Using a Docs-as-Code Approach," *AIAA SCITECH 2023 Forum*, 2023. https://doi.org/10.2514/6.2023-1126

[33] do Vale Pereira, P., Depine, F. J., and Bucher, R., "An Open-Source Python-Based Framework of Real-Time Controls as Flight Software of CubeSats," *AIAA SCITECH 2024 Forum*, 2024. https://doi.org/10.2514/6.2024-1663

[34] Liubimov, O., Turkin, I., Pavlikov, V., and Volobuyeva, L., "Agile Software Development Lifecycle and Containerization Technology for CubeSat Command and Data Handling Module Implementation," *Computation 2023*, Vol. 11, No. 9, 2023, p. 182. https://doi.org/10.3390/COMPUTATION11090182

[35] De la Vega-Martínez, M., Velázquez-García, M. C., Zavala-López, M. F., Hernández, E., Gutiérrez-Esparza, R. A., Arcos-Bravo, D. G., Medina, D., Gilardi-Velázquez, H. E., and McComas, D., "Implementation of the CFS Framework for the Development of Software in Aerospace Missions: First Application in an Undergraduate Program in Mexico," *IFAC-PapersOnLine*, Vol. 54, No. 12, 2021, pp. 88–93. https://doi.org/10.1016/j.ifacol.2021.11.014

[36] Buckner, S., Carrasquillo, C., Elosegui, M., and Bevilacqua, R., "A Novel Approach to CubeSat Flight Software Development Using Robot Operating System (ROS)," *Small Satellite Conference*, 2020. Retrieved 2 April 2023. https://digitalcommons.usu.edu/smallsat/2020/all2020/241

[37] Hishmeh, S. F., Doering, T. J., and Lumpp, J. E., "Design of Flight Software for the KySat CubeSat Bus," *IEEE Aerospace Conference*, 2009. https://doi.org/10.1109/AERO.2009.4839646

[38] Cols Margenet, M., Harris, A., and Schaub, H., "Software Architecture for Deep-Space Navigation Filter Development," *Proceedings of the International Astronautical Congress, IAC*, Vol. 22, 2017, p. 38512. Retrieved 2 April 2023. https://hanspeterschaub.info/Papers/ColsMargenet2017a.pdf

[39] Normann, M. A., and Birkeland, R., "Software Design of an Onboard Computer for a Nanosatellite," Norwegian University of Science and Technology, 2016.

[40] Eshaq, M., Al-Midfa, I., Al-Shamsi, Z., Atalla, S., Al-Mansoori, S., and Al-Ahmad, H., "Flight Software Design and Implementation for a CubeSat," *2023 Advances in Science and Engineering Technology International Conferences (ASET)*, 2023, pp. 1–6. https://doi.org/10.1109/ASET56582.2023.10180675

[41] Fitzpatrick, D. J., Rainville, N., Palo, S. E., and Woods, T., "Designing a Bare-Metal Flight Software Architecture for the Academic SWARM-EX CubeSat Constellation," *AIAA SCITECH 2024 Forum*, 2024. https://doi.org/10.2514/6.2024-1664

[42] Lamichhane, K., Kiran, M., Kannan, T., Sahay, D., Ranjith, H. G., and Sandya, S., "Embedded RTOS Implementation for Twin Nano-Satellite STUDSAT-2," *Proceedings of the 2nd IEEE International Workshop on Metrology for Aerospace, MetroAeroSpace 2015*, 2015, pp. 541–546. https://doi.org/10.1109/MetroAeroSpace.2015.7180715

[43] Monowar, M. I., Cho, M., Monowar, M. I., and Cho, M., "Over-the-Air Firmware Update for an Educational CubeSat Project," *International Review of Aerospace Engineering (IREASE)*, Vol. 14, No. 1, 2021, pp. 39–50. https://doi.org/10.15866/IREASE.V14I1.19832

[44] Gatherer, A., and Manchester, Z., "Magnetorquer-Only Attitude Control of Small Satellites Using Trajectory Optimization," *Proceedings of AAS/AIAA Astrodynamics Specialist Conference*, 2019. Retrieved 2 April 2023. https://www.ri.cmu.edu/publications/magnetorquer-only-attitude-control-of-small-satellites-using-trajectory-optimization/

[45] Marlow, W., Marinan, A., and Riesing, K., "Attitude Determination and Control Approach to Achieve Co-Located Microwave Radiometer and GPS Radio Occultation Measurements on a Nanosatellite," *2015 AAS Guidance, Navigation, and Control (GNC) Conference*, 2015. Retrieved 2 April 2023

[46] Schulte, P. Z., and Spencer, D. A., "Development of an Integrated Spacecraft Guidance, Navigation, & Control Subsystem for Automated Proximity Operations," *Acta Astronautica*, Vol. 118, 2016, pp. 168–186. https://doi.org/10.1016/J.ACTAASTRO.2015.10.010

[47] Abdelrahman, N., Annenkova, A., Ivanov, D., and Pritykin, D., "Enhancing CubeSat Active Magnetic Attitude Control Based on the Results of the Ground Tests," *28th Saint Petersburg International Conference on Integrated Navigation Systems, ICINS 2021*, 2021. https://doi.org/10.23919/ICINS43216.2021.9470849

[48] Sullivan, J., Gambone, E., Kirven, T., Pedrotty, S., and Wood, B., "Rapid Development of the Seeker Free-Flying Inspector Guidance, Navigation, and Control System," *42nd Annual AAS Guidance, Navigation, and Control Conference*, 2019. Retrieved 2 April 2023. https://ntrs.nasa.gov/api/citations/20190000520/downloads/20190000520.pdf

[49] Grace, J., Soares, L. M. P., Loe, T., and Bellardo, J., "A Low Cost Star Tracker for CubeSat Missions," *AIAA SCITECH 2022 Forum*, 2022. https://doi.org/10.2514/6.2022-0520

[50] Manyak, G., and Bellardo, J. M., "PolySat's Next Generation Avionics Design," *2011 IEEE 4th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2011, pp. 69–76. https://doi.org/10.1109/SMC-IT.2011.13

[51] Mahmood, R., Khurshid, K., and Ul Isalm, Q., "Subsystem Testing, Integration and Compliance of ICUBE-1," *2013 6th International Conference on Recent Advances in Space Technologies (RAST)*, 2013, pp. 881–886. https://doi.org/10.1109/RAST.2013.6581338

[52] Rankin, I., Rankin, K., Lloyd, S., McNeil, I., McGinnis, A., Roberts, M., Stochaj, S., de Nolfo, G., Suarez, G., Dumonthier, J., Liceaga-Indart, I., and Grant Mitchell, J., "Ionic Neutron Content Analyzer: System

Design of a Student Built 3U CubeSat," *Proceedings of the International Astronautical Congress, IAC*, 2019. Retrieved 3 April 2023

[53] Loehrlein, R. S., and Kalsch, N. J., "Lessons Learned from an Undergraduate University's First Cubesat," *Proceedings of the International Astronautical Congress, IAC*, 2019. Retrieved 3 April 2023

[54] Eshaq, M., Al-Midfa, I., Al-Shamsi, E., Al-Shamsi, Z., Atalla, S., Al-Mansoori, S., and Al-Ahmad, H., "Information Processing and Digital Communications in a Modular Satellite," *2021 4th International Conference on Signal Processing and Information Security (ICSPIS)*, 2021, pp. 72–75. https://doi.org/10.1109/ICSPIS53734.2021.9652183

[55] Ferreira, J., "ISTNanosat-1 Heart Processing and Digital Communications Unit," Technical University of Lisbon, Lisbon, 2012.

[56] Asundi, S. A., and Fitz-Coy, N. G., "Design of Command, Data and Telemetry Handling System for a Distributed Computing Architecture CubeSat," *IEEE Aerospace Conference*, 2013. https://doi.org/10.1109/AERO.2013.6496901

[57] Carreno-Luengo, H., Amezaga, A., Bolet, A., Vidal, D., Jane, J., Munoz, J. F., Olive, R., Camps, A., Carola, J., Catarino, N., Hagenfeldt, M., Palomo, P., and Cornara, S., "3CAT-2: A 6U CubeSat-Based Multi-Constellation, Dual-Polarization, and Dual-Frequency GNSS-R and GNSS-RO Experimental Mission," *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2015, pp. 5115–5118. https://doi.org/10.1109/IGARSS.2015.7326984

[58] Maldonado, C. A., Deming, J., Mosley, B. N., Morgan, K. S., McGlown, J., Nelson, A., Fernandes, P. A., Kroupa, M., Katko, K., Hehlen, M. P., Arnold, D., Barney, J., Safi, C., Pyle, M., Schultz, T., Reisenfeld, D., Skoug, R., Guider, A., Holloway, M., Morning, H., Krause, E., Sandoval, B., Beckman, D., Miller, Z., Merl, R., Graham, P. S., White, T. P., Tripp, Z., Hoose, B., Roecker, C., Klimenko, A., Dutch, R., Kaufeld, K., Cox, E., Cole, Q., Clanton, C., Bloser, P., Larsen, B. A., Fairbanks, T., George, J., Michel, J., Alpine, E. L., Kelby, C., and Abbott, B. F., "The Experiment for Space Radiation Analysis: A 12U CubeSat to Explore the Earth's Radiation Belts," *IEEE Aerospace Conference*, 2022. https://doi.org/10.1109/AERO53065.2022.9843239

[59] Spencer, D. A., Betts, B., Bellardo, J. M., Diaz, A., Plante, B., and Mansell, J. R., "The LightSail 2 Solar Sailing Technology Demonstration," *Advances in Space Research*, Vol. 67, No. 9, 2021, pp. 2878–2889. https://doi.org/10.1016/J.ASR.2020.06.029

[60] Sicsik, A., de Séréville, G., Maurice, L., Ziehlmann, P. P. D., Pinard, A., Assémat, Q., Diniz, D., El Jaïd, A., Gachod, C., Gilet, T., Girault, J. N., He, J., Hennion, L., Pauze, T., Shariatian, D., Tamrat, M., Tavant, A., Colpari, L. R., and Marmuse, F., "Detailed Design of IonSat: A Station-Keeping Mission at Altitudes Below 300km," *Proceedings of the International Astronautical Congress, IAC*, Vol. E2, 2021. Retrieved 3 April 2023

[61] Cho, D. H., Choi, W. S., Kim, M. K., Kim, J. H., Sim, E., and Kim, H. D., "High-Resolution Image and Video CubeSat (HIREV): Development of Space Technology Test Platform Using a Low-Cost CubeSat Platform," *International Journal of Aerospace Engineering*, Vol. 2019, 2019. https://doi.org/10.1155/2019/8916416

[62] Lightholder, J., Thompson, D. R., Castillo-Rogez, J., and Basset, C., "Near Earth Asteroid Scout CubeSat Science Data Retrieval Optimization Using Onboard Data Analysis," *IEEE Aerospace Conference*, 2019. https://doi.org/10.1109/AERO.2019.8742190

[63] Chiu, Y. C., Chang, L. C., Chao, C. K., Tai, T. Y., Cheng, K. L., Liu, H. T., Tsai-Lin, R., Liao, C. T., Luo, W. H., Chiu, G. P., Hou, K. J., Wang, R. Y., Gacal, G. F., Lin, P. A., Denduonghatai, S., Yu, T. R., Liu, J. Y., Chandran, A., Athreyas, K. B. N., Hari, P., Varghese, J. J., and Meftah, M., "Lessons Learned from IDEASSat: Design, Testing, on Orbit Operations, and Anomaly Analysis of a First University CubeSat Intended for Ionospheric Science," *Aerospace*, Vol. 9, No. 2, 2022, p. 110. https://doi.org/10.3390/AEROSPACE9020110

[64] Visagie, L., Steyn, W. H., Burger, H., and Malan, D. F., "Flight Results of the NSight-1 QB50 CubeSat Mission," *Advances in the Astronautical Sciences*, Vol. 163, 2018, pp. 185–198. Retrieved 3 April 2023

[65] Asundi, S. A., and Fitz-Coy, N. G., "CubeSat Mission Design Based on a Systems Engineering Approach," *IEEE Aerospace Conference*, 2013. https://doi.org/10.1109/AERO.2013.6496900

[66] Schoolcraft, J., Klesh, A., and Werne, T., "MarCO: Interplanetary Mission Development on a Cubesat Scale," *Space Operations: Contributions from the Global Community*, 2017, pp. 221–231. https://doi.org/10.1007/978-3-319-51941-8_10/FIGURES/6

[67] Latachi, I., Karim, M., and Rachidi, T., "Toward a Reusable and Fail-Safe Flight Software Architecture for Cost-Efficient Student Cubesat Missions," *Proceedings of the International Astronautical Congress, IAC*, Vol. B4, 2021. Retrieved 3 April 2023

[68] Dickinson, J. R., and George, D. E., "The High Reliability Southwest LEO Explorer (SLX-6) CubeSat Bus," *IEEE Aerospace Conference*, 2016. https://doi.org/10.1109/AERO.2016.7500943

[69] Benson, I., Kaplan, A., Flynn, J., and Katz, S., "Fault-Tolerant and Deterministic Flight-Software System for a High Performance CubeSat," *International Journal of Grid and High Performance Computing (IJGHPC)*, Vol. 9, No. 1, 2017, pp. 92–104. https://doi.org/10.4018/IJGHPC.2017010108

[70] Jackson, B., "A Robust Fault Protection Architecture for Low-Cost Nanosatellites," *IEEE Aerospace Conference*, 2014. https://doi.org/10.1109/AERO.2014.6836506

[71] Yakovyna, V., and Symets, I., "Reliability Assessment of CubeSat Nanosatellites Flight Software by High-Order Markov Chains," *Procedia Computer Science*, Vol. 192, 2021, pp. 447–456. https://doi.org/10.1016/J.PROCS.2021.08.046

[72] MacKey, R., Nikora, A., Altenbuchner, C., Bocchino, R., Sievers, M., Fesq, L., Kolcio, K. O., Litke, M. J., and Prather, M., "On-Board Model Based Fault Diagnosis for CubeSat Attitude Control Subsystem: Flight Data Results," *IEEE Aerospace Conference*, 2021. https://doi.org/10.1109/AERO50100.2021.9438342

[73] Ireland, M. L., Mendham, P., Greenland, S., Karagiannakis, P., and Hogervorst, F., "Enabling and Assuring Autonomy in Small Satellite Missions," *Sensors, Systems, and Next-Generation Satellites XXIV*, Vol. 11530, 2020, p. 1153011. https://doi.org/10.1117/12.2574612

[74] Bingham, B., and Weston, C., "System Level Hardware-in-the-Loop Testing for Cubesats," *Advances in the Astronautical Sciences*, Vol. 151, 2014, pp. 701–716. Retrieved 3 April 2023

[75] Kiesbye, J., Messmann, D., Preisinger, M., Reina, G., Nagy, D., Schummer, F., Mostad, M., Kale, T., and Langer, M., "Hardware-in-the-Loop and Software-in-the-Loop Testing of the MOVE-II CubeSat," *Aerospace*, Vol. 6, No. 12, 2019, p. 130. https://doi.org/10.3390/AEROSPACE6120130

[76] Arthurs, R., and Zoltan, A., "Software for Testing and Mitigating Radiation-Induced Effects in Commercially Available Integrated Circuits," *Proceedings of the International Astronautical Congress, IAC*, 2020. Retrieved 3 April 2023. https://static1.squarespace.com/static/5c941892049079507e8edf50/t/63293352c7a4476de329c00f/1663644505461/IAC2020__Radiation_Paper.pdf

[77] Gutierrez, T., Bergel, A., Gonzalez, C. E., Rojas, C. J., and Diaz, M. A., "Systematic Fuzz Testing Techniques on a Nanosatellite Flight Software for Agile Mission Development," *IEEE Access*, Vol. 9, 2021, pp. 114008–114021. https://doi.org/10.1109/ACCESS.2021.3104283

[78] Gutierrez, T., Bergel, A., Gonzalez, C. E., Rojas, C. J., and Diaz, M. A., "Toward Applying Fuzz Testing Techniques on the Suchai Nanosatellites Flight Software," *2020 IEEE Congreso Bienal de Argentina (ARGENCON)*, 2020, pp. 1–4. https://doi.org/10.1109/ARGENCON49523.2020.9505388

[79] Santangelo, A. D., and Skentzos, P., "Utilizing the Globalstar Network for Cubesat and Small Satellite Communications," *33rd AIAA International Communications Satellite Systems Conference and Exhibition, ICSSC 2015*, 2015. https://doi.org/10.2514/6.2015-4308

[80] Gonzalez, C. E., Bergel, A., and Diaz, M. A., "Nanosatellite Constellation Control Framework Using Evolutionary Contact Plan Design," *2021 IEEE 8th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2021, pp. 85–92. https://doi.org/10.1109/SMC-IT51442.2021.00018

[81] Harvey, R., "Flight Software and Software-Driven Approaches to Small Satellite Networks," *Handbook of Small Satellites: Technology, Design, Manufacture, Applications, Economics and Regulation*, 2020, pp. 295–329. https://doi.org/10.1007/978-3-030-36308-6_87/COVER

[82] Tipaldi, M., Legendre, C., Koopmann, O., Ferraguto, M., Wenker, R., and D'Angelo, G., "Development Strategies for the Satellite Flight Software On-Board Meteosat Third Generation," *Acta Astronautica*, Vol. 145, 2018, pp. 482–491. https://doi.org/10.1016/J.ACTAASTRO.2018.02.020

[83] Amazon Web Services, "FreeRTOS - Market Leading RTOS (Real Time Operating System) for Embedded Systems with Internet of Things Extensions," 2023. Retrieved 4 April 2023. https://www.freertos.org/

[84] Grillmayer, L., and Arnold, S., "Integrating the Cubesat Space Protocol into GSOC's Multi-Mission Environment," *Small Satellite Conference*, 2020. Retrieved 28 May 2023. https://digitalcommons.usu.edu/smallsat/2020/all2020/181

[85] McComas, D., "Increasing Flight Software Reuse with OpenSatKit," *IEEE Aerospace Conference*, 2018. https://doi.org/10.1109/AERO.2018.8396631

[86] "Flight Software Development Kit | Bright Ascension," 2023. Retrieved 12 May 2023. https://brightascension.com/products/flight-software-development-kit/

[87] Doyle, M., Gloster, A., O'toole, C., Mangan, J., Murphy, D., Dunwoody, R., Emam, M., Erkal, J., Flanaghan, J., Fontanesi, G., Rajagopalan Nair, R., Reilly, J., Salmon, L., Sherwin, D., Thompson, J., Walsh, S., De Faoite, D., Javaid, U., Mcbreen, S., Mckeown, D., O'callaghan, D., O'connor, W., Stanton, K., Ulyanov, A., Wall, R., and Hanlon, L., "Flight Software Development for the EIRSAT-1 Mission," *Proceedings of the 3rd Symposium on Space Educational Activities*, 2019, pp. 157–161. https://doi.org/10.29311/2020.39

[88] "LibreCube – Open Source Space Exploration," 2023. Retrieved 12 May 2023. https://librecube.org/
[89] European Space Agency, "NanoSat MO Framework," 2022. Retrieved 4 April 2023. https://nanosat-mo-framework.github.io/
[90] "PyCubed," GitHub, 2024. Retrieved 15 December 2024. https://github.com/pycubed
[91] Miranda, D. J. F., Ferreira, M., Kucinskis, F., and McComas, D., "A Comparative Survey on Flight Software Frameworks for 'New Space' Nanosatellite Missions," *Journal of Aerospace Technology and Management*, Vol. 11, 2019, p. e4619. https://doi.org/10.5028/JATM.V11.1081
[92] STMicroelectronics, "UM1721 User Manual Developing Applications on STM32Cube™ with FatFs," 2019.
[93] Beningo, J., "A Review of Watchdog Architectures and Their Application to Cubesats," April 2010.
[94] STMicroelectronics, "STM32CubeIDE - Integrated Development Environment for STM32 - STMicroelectronics," 2023. Retrieved 6 April 2023. https://www.st.com/en/development-tools/stm32cubeide.html