Runtime Verification Triggers Real-time, Autonomous Fault Recovery on the CySat-I*

Alexis Aurandt[0000−0003−2008−673X], Phillip H. Jones[0000−0002−8220−7552], and Kristin Yvonne Rozier[0000−0002−6718−2828]

Iowa State University, Ames, IA 50010, USA
{aurandt, phjones, kyrozier}@iastate.edu

Abstract. CubeSats are low-cost platforms that are popular for conducting space-borne experiments, however they are known to have high failure rates (∼25% failure rate). In order to improve the likelihood of success of Iowa State University's first CubeSat (CySat-I), we integrate Runtime Verification (RV) on the CySat-I to allow for fault detection at runtime. Although CubeSats have been previously identified as a possible target for RV, this is the first time that a RV engine has been deployed on a CubeSat. We utilize the R2U2 runtime verification engine due to its low overhead; we embed R2U2 directly on the On-Board Computer (OBC) to monitor the current state of the CySat-I. R2U2 continuously monitors the different subsystems on the CySat-I, and R2U2's fault detection triggers pre-defined fault recovery strategies. Since the Electrical Power System (EPS) is a common source of failure, we specifically focus on this subsystem. We design a list of twenty-two specifications from English requirements corresponding to the EPS and translate them into Mission-time Linear Temporal Logic (MLTL). We perform mock launches on Earth with external fault injection to illustrate that R2U2 successfully reasons about faults and the CySat-I effectively performs fault recovery. We demonstrate that the CySat-I can successfully recover from eight unique EPS faults at runtime in a timely manner with no errors. During our mock launches, R2U2 discovered a potential error in the manufacturer's firmware related to the EPS's under-voltage event monitoring, and this led to a more in-depth investigation of the error by the manufacturers.

Keywords: Online Runtime Verification · R2U2 · Temporal Logic · Formal Specification · Fault Recovery · CubeSat

1 Introduction

Since the first CubeSat was launched in 2003, the number of CubeSats launched each year has increased exponentially, and as of December 2021, a total of 1,663 CubeSats have been launched [12, 24, 27]. This exponential growth in CubeSats is due to their low-cost and capability for fast development. CubeSats allow for both academic institutions and commercial sectors to gain easy space access with limited resources and time requirements. With the increase in popularity of CubeSats, the technology and research behind CubeSats has also advanced. This has lead to a decrease in failure rate over the years, but the failure rate is still troubling at approximately 25% [27].

* Supported by NSF:CPS Award 2038903. Reproducibility artifacts available at *http://temporallogic.org/research/CySat-NFM22*.

Failure within CubeSats is common due to a lack of proper integration and system testing before launching [14, 24, 25]. Furthermore, universities tend to have higher failure rates than their commercial counterparts due to more constrained resources and development schedules [13, 14, 24, 26]. If more time is dedicated to integration and system testing, most causes of failure could be discovered before the satellite is ever launched. Since fast development time is one of the attributes that make CubeSats attractive, most CubeSats will never have fully exhaustive integration and system testing before becoming spaceborne. Runtime Verification (RV) provides a unique mitigation. RV adds an independent check for real-time triggering of appropriate fault recovery strategies. Additionally, RV is a useful tool for finding errors in the system during testing on Earth; it provides different coverage than traditional system testing to allow for finding difficult errors with less effort.

Most CubeSat failures originate in the Electrical Power System (EPS), Attitude Determination and Control System (ADCS), and the communications system [2,13,24]. These subsystems are mission-critical; if any of these subsystems fail, the entire satellite experiences failure. A recent study formally verified a CubeSat's ADCS at design time to provide runtime assurance [8]. Also, [15] provides a case study of deploying runtime verification on a simulated CubeSat communications system. We focus on the EPS as it has never been evaluated for formal verification and it contributes to approximately one-third of CubeSat failures [13].

The CySat-I's Onboard Computer (OBC) has strict real-time constraints as it is responsible for commanding and monitoring all the other subsystems. The OBC is also restricted to 2MB of program memory [6]. The Realizable, Responsive, Unobtrusive Unit (R2U2) is a unique RV engine in that it requires little overhead and has a fast response time [18, 21]. In addition, R2U2 has been previously deployed on several resource-constrained hard real-time systems [3, 9, 10]. The CySat-I team selected R2U2 as the RV engine due to its configurability for resource-constraints, real-time verdict streaming, and proven unobtrusive monitoring of other real-time systems, e.g., [4, 10, 19]. Our implementation of fault recovery with the aid of R2U2 is currently planned to launch onboard the CySat-I in October 2022.

We contribute (1) elicitation of twenty-two realistic EPS specifications from English requirements translated into Mission-time Linear Temporal Logic (MLTL), (2) external fault injection to demonstrate that the CySat-I autonomously recovers from eight unique EPS faults in real-time, and (3) firmware error discovery during testing with the help of R2U2. Our categorization technique for the elicitation of EPS specifications is generalizable for application to other mission-critical systems. The remainder of the paper is organized as follows. Section 2 outlines the CySat-I architecture. Section 3 details the implementation of R2U2 on the CySat-I. Section 4 describes the development of the twenty-two specifications. Our mock launch setup with external fault injections appears in Section 5. We analyze the mock launch results and plot data revealing a firmware error in Section 6. In Section 7, we draw conclusions and explore future plans.

2 System Description

The CySat-I is a 3U CubeSat (10cm x 10cm x 30cm) that was designed by students at Iowa State University through the Aerospace Department's Make to Innovate program [17]. The CySat-I is composed of a mix of commercial off-the-shelf (COTS) and

custom components interconnected in a stack by PC/104 connectors as shown in Figure 1. The OBC, EPS, and UHF are COTS components from Endurosat. The OBC hosts a STM32F427 ARM Cortex processor [6] serving as the brain of the satellite; it is responsible for coordinating the other subsystems. The EPS manages how the solar panels charge the batteries and manages when different power buses and subsystems are powered on/off. The Ultra-High Frequency Radio (UHF) is responsible for deploying the antenna and communicating with the ground station. The ADCS, a COTS component from CubeSpace, is responsible for orientating the satellite towards Earth. The boost board is a custom component that amplifies the 5 volts produced by the EPS to the 7.4 volts required by the ADCS. The payload is another custom component, and it consists of a FPGA that hosts a Linux-based software defined radio (SDR). The payload's SDR reads measurements from an array of low-noise amplifiers (LNAs) to measure soil moisture on Earth [16].

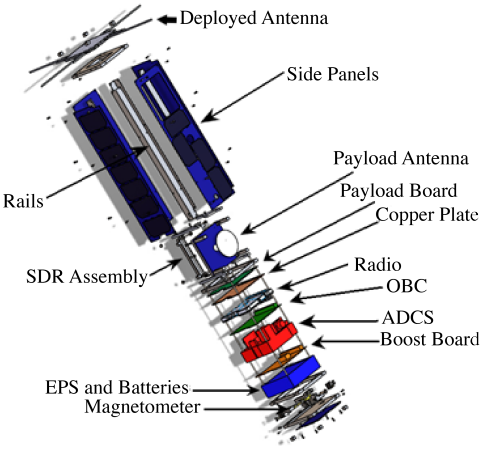

Fig. 1. Exploded view of the CySat-I and all of its components

3 Implementation

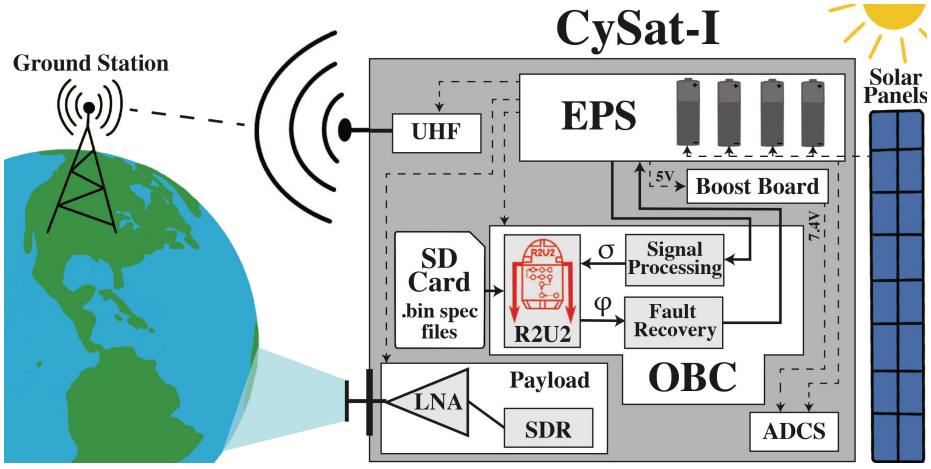

Fig. 2. R2U2 Integration. Specification binary files are loaded into R2U2 from a SD card. EPS data is gathered and processed during "Signal Processing", and this outputs the signals (σ) that are inputted into R2U2. Based on the loaded specifications, R2U2 supplies the output verdict (φ) for each of the inputs (σ). The output verdicts (φ) are used by the OBC's "Fault Recovery" to autonomously trigger the applicable EPS mitigation action. The power supply lines are indicated by dashed lines.

We deploy R2U2 directly onto the OBC of the CySat-I using the STM32CubeIDE [22]. The C version of R2U2 requires 16KB of the OBC's 2MB program memory

(0.8%), which leaves plenty of room for the CySat-I mission software (180KB). We translated the CySat-I mission requirements from the Endurosat EPS user manual [5, 7] and the CySat-I concept of operations manual [11] into MLTL specifications. MLTL concisely captures the strict temporal mission requirements and is a native language of R2U2 [18, 21]. We compiled the specifications and loaded the specification binaries onto the OBC's SD card. The OBC loads the specifications once into R2U2 upon initial boot-up. FreeRTOS, a real-time operating system, manages the OBC's tasks [1]. FreeRTOS launches a five second periodic task that will gather and process status information from the EPS, input the signals into R2U2, and store the `false` output verdicts produced by R2U2 into an array. The OBC evaluates this array, and whenever a `false` output verdict occurs (i.e., a specification is violated), a predefined mitigation strategy is triggered. Figure 2 illustrates this integration of R2U2 into the CySat-I.

4 Runtime Specification Development

We elicit specifications according to the categorization scheme presented in [20] and used, e.g., in [3], including patterns for "operating range," "rate of change," "control sequence," and "physical model relationship" specifications. [1]

Satellite power up. During the first thirty minutes after launch from the International Space Station (ISS), it is strictly required by the ISS that a CubeSat can only have its EPS and OBC subsystems powered on. Specification (1) captures this requirement. Since the FreeRTOS task that runs R2U2 is launched every five seconds, the $G_{[0,360]}$ part of this specification covers the first thirty minutes of the mission (i.e., 5 seconds * 360 = 30 minutes). During this time, all power buses (except for the 3.3 volt bus required for the OBC) and all enable signals must be in the off/disabled state.

$$\begin{aligned} G_{[0,360]}\{\neg 5V\_Bus\_Enabled \wedge \neg LUP\_5V\_Bus\_Enabled \wedge \\ \neg LUP\_3.3V\_Bus\_Enabled \wedge \neg ADCS\_Active \\ \wedge \neg Payload\_Enabled \wedge \neg UHF\_Enabled \wedge \\ \neg Boost\_Board\_Enabled\} \end{aligned} \quad (1)$$

Power bus requirement. Specification (2) captures that any time the UHF is enabled at least thirty minutes after launch, then the latch-up protected (LUP) 3.3 volt bus must also be enabled. The LUP 3.3 volt bus is a UHF input required for proper operation. The $G_{[360,M]}$ part of the specification established that this specification must hold from thirty minutes after launch till the end of the mission indicated by M. Corresponding requirements for the boost board and payload form specifications (3) and (4).

$$G_{[360,M]}\{UHF\_Enabled \rightarrow LUP\_3.3V\_Bus\_Enabled\} \quad (2)$$

$$G_{[360,M]}\{Boost\_Booard\_Enabled \rightarrow 5V\_Bus\_Enabled\} \quad (3)$$

$$G_{[360,M]}\{Payload\_Enabled \rightarrow 5V\_Bus\_Enabled\} \quad (4)$$

[1] All twenty-two specifications with categorization appear here: *http://temporallogic.org/research/CySat-NFM22*.

Under-voltage event. Whenever the EPS's output power buses fall below a given voltage threshold, the EPS's lifetime *under-voltage event* counter increments [5]. Specification (5) uses this information to compare the current value (value at mission time i) of this status value to its previous value (value at mission time $i-1$). If these are not equal, then an *under-voltage event* has occurred. In this specification, $G_{[0,M]}$ checks that the requirement holds from the beginning to the end of the mission.

$$G_{[0,M]}\{Num\_Under\_Voltage_i == Num\_Under\_Voltage_{i-1}\} \tag{5}$$

I2C communication. The OBC communicates with the EPS over an I2C bus interface. It was documented in [2] that I2C communication errors can cause EPS failure. To mitigate this mode of failure, we instrumented the OBC's I2C driver to report and accumulate communication errors (e.g., NACKs, transaction timeouts). Specification (6) detects whenever a new I2C error occurs. If the total number of I2C errors at the current mission-time does not equal the total number of errors at the previous mission time, then this specification does not hold. In the event that R2U2 detects the failure of this specification, it triggers the fault mitigation action of resetting the I2C bus.

$$G_{[0,M]}\{Num\_I2C\_Errors_i == Num\_I2C\_Errors_{i-1}\} \tag{6}$$

5 Evaluation Methodology

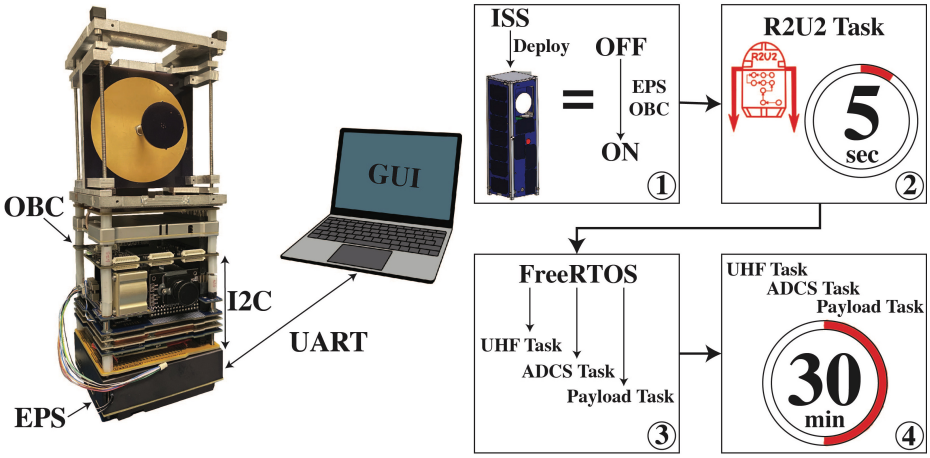

Fig. 3. Mock Launch. Left: The physical CySat-I PC/104 stack without the external structures (e.g., solar panels) and its setup during the mock launches. Right: Mock launch sequence.

We conduct mock launches to evaluate the correct implementation of our specifications, deployment of R2U2 within the CySat-I, and implementation of our fault recovery mechanisms. Within the CySat-I PC/104 stack, the EPS communicates with the OBC via an I2C bus. The EPS also has a UART connection available over a USB port. Endurosat provides a GUI that can interact with the EPS's UART interface while the EPS is plugged into the PC/104 stack. This setup is depicted in Figure 3. We leverage this GUI during mock launches to inject power bus faults by turning buses on/off and subsystem enable faults by enabling/disabling different subsystems. As shown in

Figure 3, a mock launch consists of: 1) powering on the EPS and OBC (i.e., emulating the CySat-I being launched from the ISS), 2) FreeRTOS on the OBC starting the R2U2 task that runs every five seconds, 3) FreeRTOS starting simplified tasks for the other subsystems, and 4) all subsystem tasks waiting thirty minutes before starting modified operation. We record and analyze the input status signals of R2U2 and the output verdicts generated by R2U2 during the mock launch fault-injection campaigns. These logs allow us to determine if faults are being detected as expected and if fault mitigation strategies are being appropriately triggered.

6 Results and Analysis

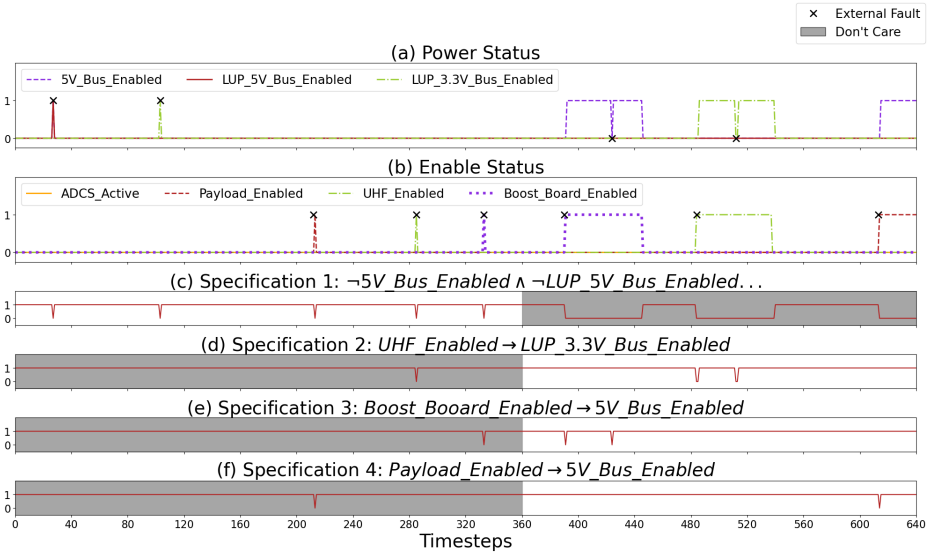

Fig. 4. EPS Fault Recovery. (a) The power status of the 5 volt, LUP 5 volt, and LUP 3.3 volt buses. (b) The enable status of the ADCS, payload, UHF, and boost board. An 'X' marker indicates an injection of an external fault. (c) (d) (e) (f) Output from R2U2 correctly determining the current state of specification (1), (2), (3), and (4) respectively. A shaded region indicates a time range where the OBC does not care about the output of R2U2 within its fault recovery.

R2U2 is a stream-based RV engine that reevaluates specifications at each time step creating an implicit global operator. Therefore, we reduce our specifications that we instruct R2U2 to reason over as depicted in Figure 4 and 5. Recall that specification (1) is only applicable for the first thirty minutes after launch, specifications (2), (3), and (4) are only applicable after the first thirty minutes, and specifications (5) and (6) are always applicable. In order to apply a specification for a certain time interval, the OBC monitors the current time step of R2U2. If not within the applicable time interval for a specification, the OBC does not care what R2U2 is outputting and does not apply a mitigation action, which is indicated by the shaded region in Figure 4.

Figure 4 illustrates an approximately hour-long mock launch with fault recovery for four unique specification faults (i.e., specification (1), (2), (3), and (4)).[2] Within the first thirty minutes, none of the plotted power buses or subsystem enables are allowed

[2] All eight specification faults appear here: *http://temporallogic.org/research/CySat-NFM22*.

to be enabled. During this period, we inject power bus and subsystem enable faults. Each time a fault is injected, two actions are observed: 1) R2U2 indicates a fault by providing a `false` verdict for specification (1), and 2) our fault recovery mechanism is triggered shown by the violating enable being disabled autonomously by the next time step. After the initial thirty minutes, we inject faults that either enable a subsystem before its required power bus is powered on or we disable a power bus while its corresponding subsystem is still enabled. In both cases, a mitigation strategy enabled the appropriate power bus by the next time step. While the time steps observed by R2U2 are five seconds, the response time of our fault recovery (i.e., time from fault detection to correction) is approximately 7 ms.

Figure 5 depicts the discovery of an error within the EPS firmware, which provides a real-world example of the benefit of using RV during testing. During a mock launch, R2U2 detected the number of *under-voltage events* changing. Upon closer examination, the value spikes from a value of ten (the expected value during this mock launch) to a value of 2308 briefly before returning back to a value of ten. After discovering this erroneous behavior with the EPS's firmware, we contacted the manufacturer who is currently investigating the issue.

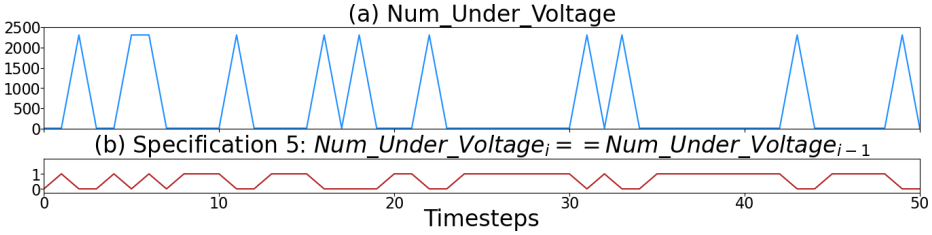

Fig. 5. Potential EPS Firmware Error. (a) The value of the *under-voltage event* counter. (b) Output from R2U2 correctly determining the current state of specification (5) to indicate a change in the *under-voltage event* counter.

7 Conclusion

In order to increase the CySat-I's chance for having a successful mission, we deployed R2U2 on the CySat-I to trigger fault recovery and monitor for errors during testing. R2U2 was able to successfully reason over twenty-two MLTL specifications and detect faults in real-time. R2U2 and our fault recovery mechanisms will ensure that several faults that could occur during the CySat-I's mission can be successfully recovered from. Additionally, if R2U2 had not been deployed on the CySat-I, we would have never uncovered the EPS firmware bug concerning the *under-voltage event* counter. The ability to perform fault recovery in real-time during the mission of the CySat-I is advantageous for the other mission-critical subsystems onboard (e.g., the ADCS and UHF); appropriate fault recovery for these subsystems can reduce the failure rate of future CubeSats. In future work, the R2U2 engine can trigger appropriate fault mitigation strategies for all mission-critical subsystems of a CubeSat, and RV can continue to be explored for CubeSat testing on Earth to assist in discovering elusive errors. We are also pursuing to publish our twenty-two EPS specifications as a benchmark to a public database (e.g., StarExec [23]).

References

1. Amazon Web Services: The FreeRTOS™ Reference Manual (2017)
2. Bouwmeester, J., Langer, M., Gill, E.: Survey on the implementation and reliability of cubesat electrical bus interfaces. In: CEAS Space Journal, vol. 9, pp. 163–173 (2017). https://doi.org/10.1007/s12567-016-0138-0
3. Cauwels, M., Hammer, A., Hertz, B., Jones, P., Rozier, K.Y.: Integrating Runtime Verification into an Automated UAS Traffic Management System, pp. 340–357 (09 2020). https://doi.org/10.1007/978-3-030-59155-7_26
4. Dabney, J.B., Badger, J.M., Rajagopal, P.: Adding a verification view for an autonomous real-time system architecture. In: Proceedings of SciTech Forum. p. Online. 2021-0566, AIAA (January 2021). https://doi.org/https://doi.org/10.2514/6.2021-0566
5. Endurosat: Electrical Power System (EPS I & EPS I Plus) - I2C Protocol User Manual (2019)
6. Endurosat: Onboard Computer (OBC) Type II - User Manual (2019)
7. Endurosat: Electrical Power System (EPS I & EPS I Plus) User Manual (2020)
8. Gross, K.H., Clark, M., Hoffman, J.A., Fifarek, A., Rattan, K., Swenson, E., Whalen, M., Wagner, L.: Formally verified run time assurance architecture of a 6u cubesat attitude control system. In: AIAA Infotech Aerospace, p. 0222 (2016)
9. Hertz, B., Luppen, Z., Rozier, K.Y.: Integrating runtime verification into a sounding rocket control system. In: NASA Formal Methods Symposium. pp. 151–159. Springer (2021). https://doi.org/10.1007/978-3-030-76384-8_10
10. Kempa, B., Zhang, P., Jones, P.H., Zambreno, J., Rozier, K.Y.: Embedding online runtime verification for fault disambiguation on robonaut2. In: Bertrand, N., Jansen, N. (eds.) Formal Modeling and Analysis of Timed Systems. pp. 196–214. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-57628-8_12
11. Kilcoin, M., Kempa, B., Goldenberg, J., Nelson, M., Gonzalez-Torres, T.: Cysat-1 concept of operations (2020), *https://iastate.box.com/s/zf6xbwwc3jb9hwshc6hc52evx2e60s13*
12. Kulu, E.: Nanosatellite & cubesat database, *https://www.nanosats.eu/database*
13. Langer, M., Bouwmeester, J.: Reliability of cubesats - statistical data, developers' belief, and the way forward. In: Proceedings of the 30th Annual AIAA/USU Conference on Small Satellites (2016)
14. Langer, M., Weisgerber, M., Bouwmeester, J., Hoehn, A.: A reliability estimation tool for reducing infant mortality in cubesat missions. In: 2017 IEEE Aerospace Conference (2017). https://doi.org/10.1109/AERO.2017.7943598
15. Luppen, Z.A., Lee, D.Y., Rozier, K.Y.: A case study in formal specifications and runtime verification of a cubesat communications system. In: AIAA SciTech Forum (2021). https://doi.org/10.2514/6.2021-0997
16. Nelson, M.E.: Implementation and evaluation of a software defined radio based radiometer. Master's thesis (2016)
17. Nelson, M.E., Lee, D.Y., Kilcoin, M., Gordon, L., Brown, W.: Preparing cysat-1: A look at iowa state university's first cubesat. In: Proceedings of the 34th Annual Small Satellite Conference (2020)
18. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proceedings of the 20th International Conference on Tools and Algorithms in Construction and Analysis of Systems (TACAS). vol. 2413, pp. 357–372. Springer-Verlag, Lecture Notes in Computer Science (LNCS) (2014)
19. Rozier, K.Y.: R2U2 in Space: System and Software Health Management for Small Satellites. In: Spacecraft Flight Software Workshop (FSW) (December 2016), *https://www.youtube.com/watch?v=OAgQFuEGSi8*

20. Rozier, K.Y.: Specification: The biggest bottleneck in formal methods and autonomy. In: Proceedings of 8th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2016). LNCS, vol. 9971, pp. 1–19. Springer-Verlag, Toronto, ON, Canada (July 2016)
21. Rozier, K.Y., Schumann, J.: R2U2: Tool Overview. In: RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools. Kalpa Publications in Computing, vol. 3, pp. 138–156. EasyChair (2017). https://doi.org/10.29007/5pch
22. STMicroelectronics: STM32CubeIDE User Manual (2020)
23. Stump, A., Sutcliffe, G., Tinelli, C.: Starexec: A cross-community infrastructure for logic solving. In: International joint conference on automated reasoning. pp. 367–373. Springer (2014)
24. Swartwout, M.A.: The first one hundred cubesats: A statistical look (2013)
25. Venturini, C., Braun, B., Hinkley, D., Berg, G.: Improving mission success of cubesats. In: Proceedings of the 32nd Annual AIAA/USU Conference on Small Satellites (2018)
26. Venturini, C.C.: 8 steps improving small set mission success, *https://aerospace.org/article/8-steps-improving-small-sat-mission-success*
27. Villela, T., Costa, C.A., Brandão, Alessandra, M., Bueno, F.T., Leonardi, R.: Towards the thousandth cubesat: A statistical overview. In: International Journal of Aerospace Engineering, vol. 2019 (2019). https://doi.org/10.1155/2019/5063145