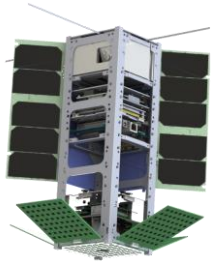
Creating Reliable Software Systems for the DORA CubeSat

Zachary Hoffmann, Judd Bowman, Daniel Jacobs
Arizona State University Low Frequency Cosmology Lab

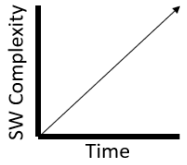
The DORA CubeSat

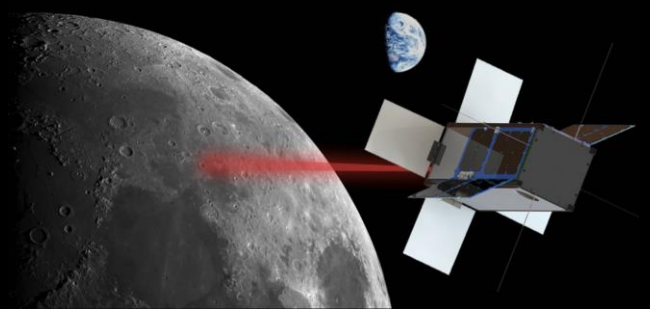
The Deployable Optical Receiver Aperture (DORA) CubeSat is a 3U CubeSat that seeks to demonstrate data rates of 1 Gbps over distances of thousands. The DORA technology presents an easy to accommodate optical communications solution for smallsats, which previously was limited by the high pointing accuracy requirement for traditional optical communication systems. We believe this technology to be best suited for surface to orbit communications and the crosslink between small spacecraft, including those in smallsat constellations/swarms.



The Software Problem

- 41.3% failure rate for SmallSats from 2000-2016 [1]
- SmallSats often developed in flexible environments [2]
- Teams face high personnel turnover rates and Inexperience problems [2]
- Complexity of code is sharply increasing with time [3]
- Higher Complexity = Higher Risk of Failure
- Each line of code is a potential defect
- Virtually impossible to achieve complete code coverage for complex code bases
- Testing can be limited by equipment available
- COTS Architecture vs Custom Architecture
- **How do we create reliable flight software?**





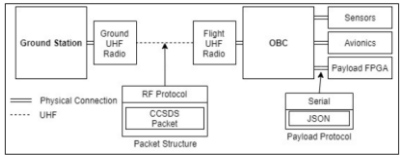
Areas of Focus and Accompanying Practices

- Error Handling
- Concurrency
- Memory

Flight Software Best Practices

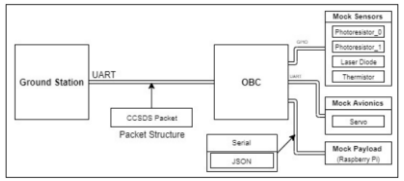
1) No dynamic memory allocation of any kind
2) Learn all the forms of dynamic memory allocation
3) Handle all shared resources accordingly
4) Treat all potentially shared resources as shared resources
5) Run concurrency analysis tests and create a concurrency model
6) Maintain control of the software at all times
7) Promote use of fault containment
8) Log all off nominal conditions in an organized manner
9) Code as if you expect the software to be broken (human error)
10) Expect off nominal conditions to occur

Testing Environments (For Code Coverage)



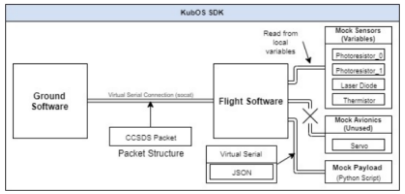
FlatSat Configuration

- General structure and interactions defined
- Exact hardware not known yet/not available
- Interfaces at a software level clearly defined
- Currently under development
- Used for final software validation



Simulated Hardware-In-The-Loop (HITL)

- Uses flight like hardware to simulate HW interactions
- Learning opportunity for working with HW
- Replicates basic FlatSat configuration
- Simple dev environment
- Used for basic testing and can be used for architecture level testing



Processor-In-The-Loop (PITL)

- Built within KubOS (chosen OS) SDK
- HW interactions simulated with noops or rnd data
- General structure preserved
- Virtual UART ports used for interfacing consistency
- Used for basic app development

Testing Approach

Test cases can be derived from the list of best practices and in doing so will draw focus to major areas of concern for flight software—Error Handling, Concurrency, and Memory. Areas of concern were chosen based on the risk they pose to the satellite if handled improperly with the three chosen areas posing the highest risk of mission failure due to the nature of the respective faults produced. With these areas in mind, each test case becomes an opportunity to look at the software's behavior from multiple perspectives. They serve as a reminder to check all aspects of the software's performance, not just the functionality under test.

In short: New functionality is tested for function, error handling ability, concurrency risk, and memory management ability.

Test Cases Example

Primary rule being tested against (number from best practices list)

Secondary rules covered by test

Test Name	Motivating Rule	Rules Covered	Value Provided
Isolated Crashes	7	1, 6, 9, 10	Design Validation
Safe-mode Triggering	10	1, 6, 8	Error handling core function
Boot Recovery	10	1, 6	Off-nominal condition coverage
Out Of Limits Events	10	1, 6, 8	Off-nominal condition coverage, logging capabilities
Mode Swapping	5	1, 9, 10	Unsafe mode swapping discovery
Periodic Tasks	5	1, 3, 4	Attention toward concurrency problems for periodic tasks

Benefit from doing test clearly stated

References

[1] Jacklin, S. A. (2018). (tech.). Small-Satellite Mission Failure Rates (pp. 1–46). Moffett Field, CA: NASA Ames Research Center.
[2] Gonzalez, C. E., Rojas, C. J., Bergel, A., & Diaz, M. A. (2019). An Architecture-Tracking Approach to Evaluate a Modular and Extensible Flight Software for CubeSat Nanosatellites. IEEE Access, 7, 126409–126429. https://doi.org/10.1109/access.2019.2927931
[3] West, A. (2009). (tech.). (D. L. Dvorak, Ed.) NASA Study on Flight Software Complexity (pp. 1–264). Pasadena, CA: Jet Propulsion Laboratory.

Point of Contact: Zachary Hoffmann zchoffma@asu.edu

Arizona State University

Jet Propulsion Laboratory
California Institute of Technology