Software: The Overlooked Glue that Holds CubeSats Together

Dr. John M. Bellardo

About Me

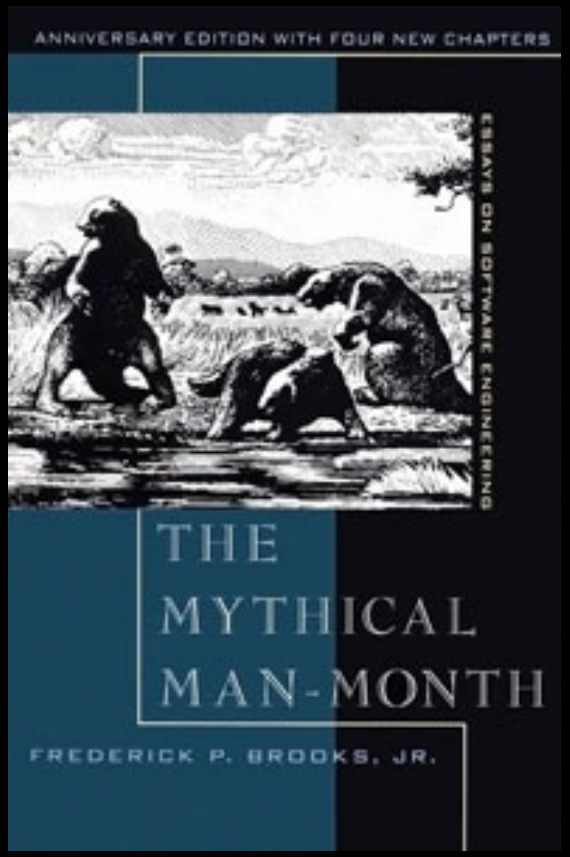
- Director of Cal Poly's CubeSat Laboratory
  - Working with CubeSats for 10+ years
  - Involved in 15+ launched CubeSat missions, including 7 in the past 12 months
  - Help maintain the CubeSat Design Standard
  - Host the Spring Developer's Workshop at Cal Poly
- Professor of Computer Science and Software Engineering at Cal Poly San Luis Obispo
- Doctorate in Computer Science and Engineering from UC San Diego

Why Software

- All CubeSats require software
- Reasons for placing additional functionality in software
  - Favorable power / volume / mass tradeoffs
  - Risk profile of CubeSat missions enables more sophisticated software
  - Enables advanced features, e.g., Artificial Intelligence

Software Challenges

- Large software projects are non-intuitive
  - PolySat has ~200k lines of in-house code, in addition to Linux
  - Large amount of custom tooling
- Tendency to get caught up with hardware compatibility, not software compatibility
- Software lacks the intuitiveness found in other areas of the spacecraft design
- Software has a bad reputation for being behind schedule and over budget

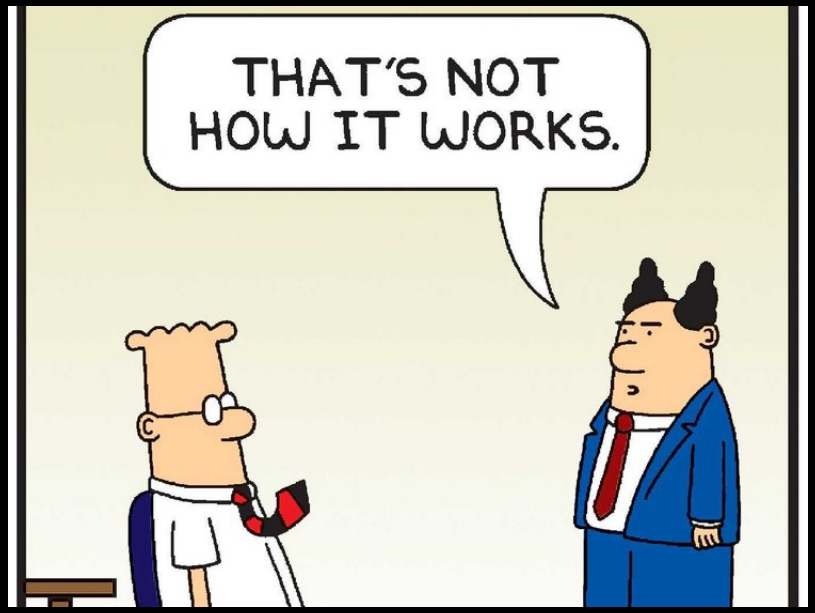


Best Practices

- Software challenges are not limited to CubeSats
- Look to the software engineering community for tools and solutions
  - Try to avoid traditional aerospace specific approaches
- Risk profile enables use of best practices that have been shown to work well on large terrestrial projects, despite lack of flight heritage

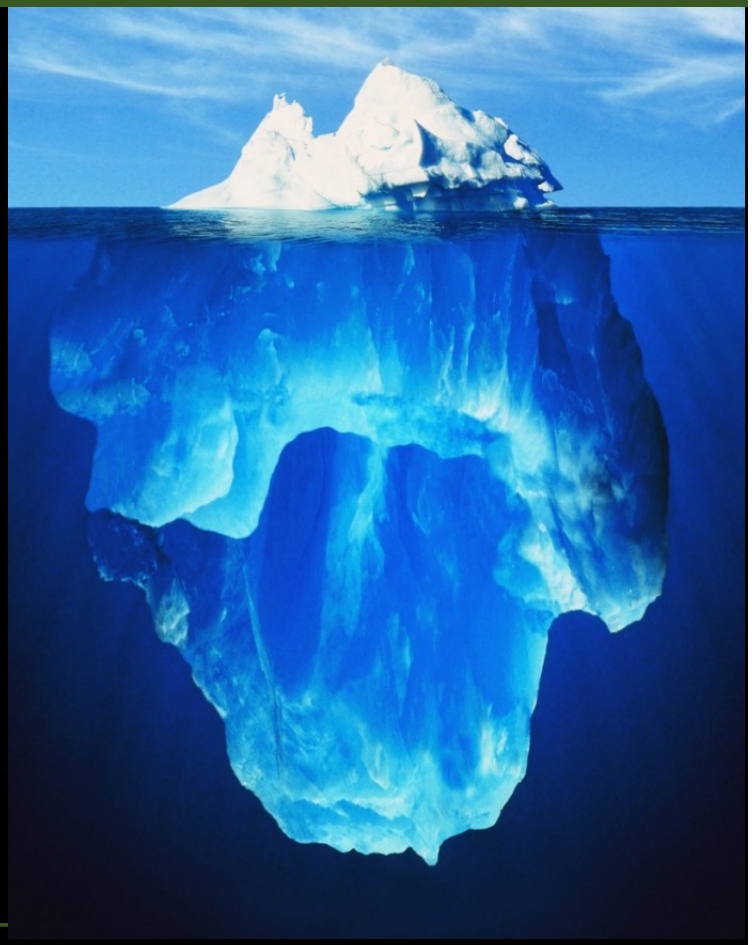
Software Managers

- Strive to find managers with formal software background to manage software
  - Training and/or experience gives them much better intuition
  - Useful in determining when there is a problem vs. development taking longer than anticipated

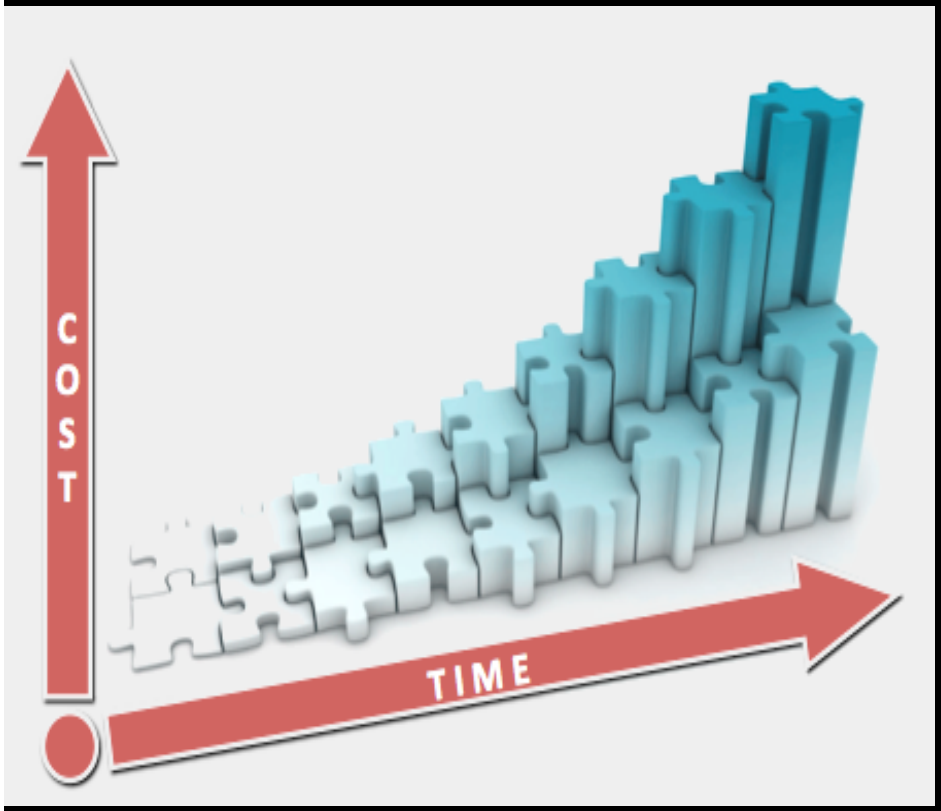


Plan for Software

- Include software team members in all your trade studies and design decisions
- It can be difficult for people inexperienced in software development to estimate time needed to support a design decision
- Example: Camera Drivers
  - Some camera vendors have robust tools and documentation on how to configure the imager's settings
    - Perhaps 100 hours of development and testing
  - Some vendor support is so poor people resort to guess-and-check techniques
    - 1000+ hours



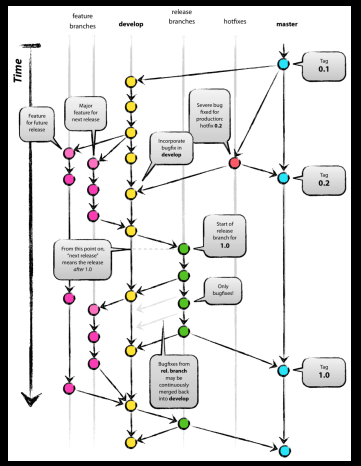
Schedule



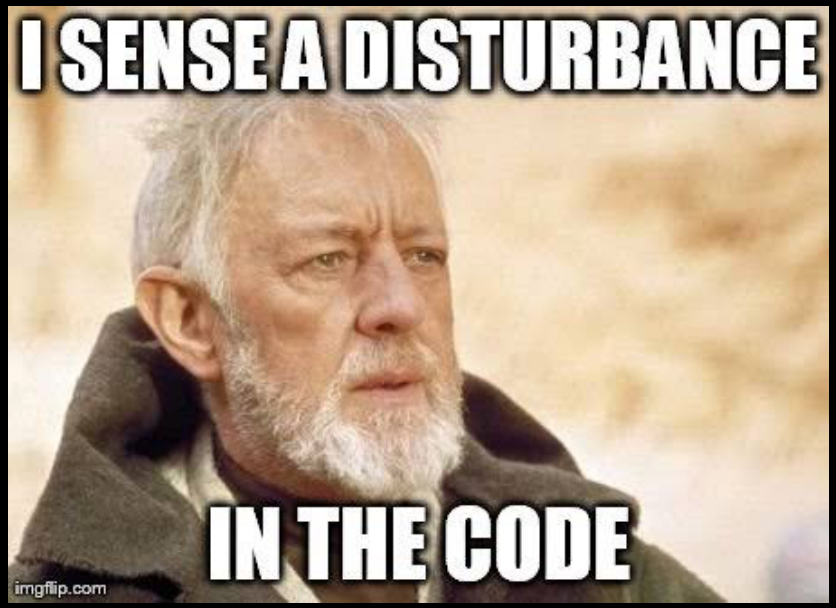
- Expect software schedules to take 3x more than your original estimate
- Move software testing as early in your schedule as possible
  - Tendency to wait until flight hardware is available
  - Inevitable slips in hardware readiness greatly impact software testing
- Look to create infrastructure necessary for early software testing
  - Prioritize prototype hardware the software team can use
  - Leverage component specific development boards
  - Have enough copies of flight hardware that the software team always has access

Revision Control

- Use strong revision control from the beginning of development
  - Git, svn are common in the development community
  - Force team members to get through the learning curve
- Use the revision control system as it was intended
  - Frequent commits
  - Branches for exploratory or independent work
  - Frequent pushes to the server
- Tag / mark all builds of flight software for full traceability



Code Reviews



- All code should be reviewed prior to being accepted by the project
- Small changes can be reviewed offline
- Larger changes require multi-hour meetings
- Human nature tends to consider this a poor use of productive time, but it is necessary to ensure higher quality software

Collaboration Tools

- Use software specific collaboration tools for software development
- Most combine revision control, code review, continuous integration, documentation, issue tracking, and more
  - Github, Atlassian, gitlab, etc
- The tools are not effective if team members don't use them
  - Help your team get through the learning curve

Documentation

- Strive to write documentation at the same time as the code
- Make sure the documentation requirements are reasonable
  - E.g., Don't institute an "every line needs a comment" policy
- Review documentation during the code review, and only accept the code when the documentation is acceptable
- For larger teams, consider involving someone whose primary role is assisting other developers with documentation

Knowledge Transfer

- Create opportunities for knowledge transfer outside of written documentation
  - Weekly seminars, both deep-dive and overview
  - In-person code reviews
  - Group discussions of architectural decisions prior to implementation

Manual Testing

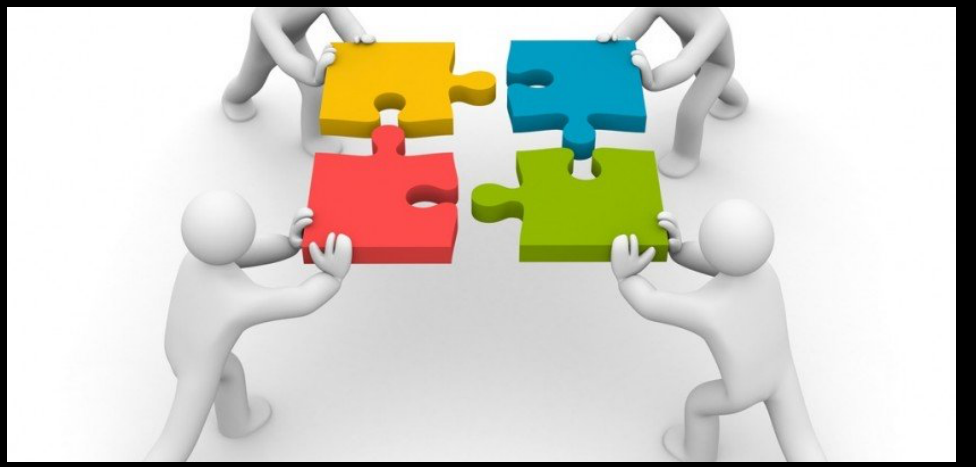
- Understand that manual testing is exceptionally ineffective for software
  - Most software bugs are found in edge cases, not the common case
  - Manual testing tends to focus on the common case because the testing itself is personnel constrained
  - Know this spot check doesn't really provide any assurance of code performance
- A test showing your antenna deploys on time uses software, but is primarily testing the integration of the hardware and software, not that the software works
- Limit testing / debugging to use commands available on orbit

Unit Testing

- Use a unit testing tool / framework
- Write unit tests!
- Require unit tests prior to code reviews
- Review unit tests, expected coverage, etc., during code reviews
- Pass all tests prior to accepting a code change
- When fixing a bug, write a test that teases out the bug prior to fixing the code
- Keep records of testing results

Integrated Testing

- Most large software projects are composed of many smaller modules with well-defined interfaces
  - Unit testing should include validation of interface functionality
- Partial and full integrated testing validates overall system behavior
- Ideally performed automatically
  - What about external input?
  - How do I test without being on orbit? How do I test on the hardware?
- Takes time to develop good integrated test framework
- Normally more code than what you are actually testing

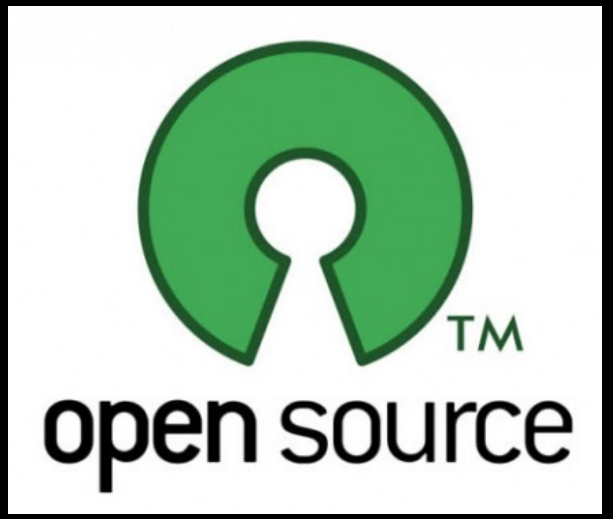


Continuous Integration

- Continuous Integration (CI) runs unit and integration tests automatically as code is committed
- Removes some of the time burden from developers
- Typically supported by revision control tools
- Can serve as a gate for accepting code changes
- Requires setup time and learning curve

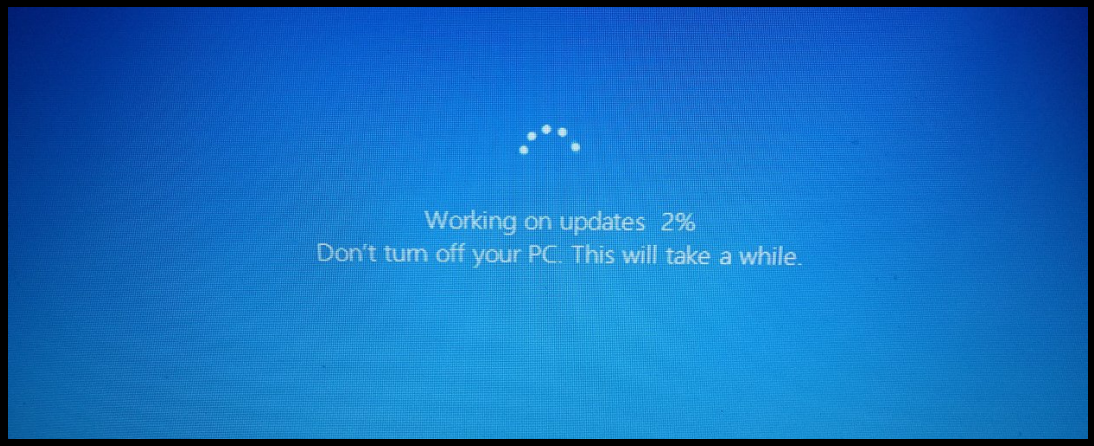
3rd Party Code

- Don't be afraid to use 3rd party code (e.g., open source, etc.)
  - Can save development, testing time
  - Common 3rd party code (e.g., Linux) has many more accumulated hours of operation than anything you will develop
  - Most performance characteristics are well understood
- It is typically faster to customize 3rd party code than develop it yourself from scratch
  - Be cognizant of not-invented-here syndrome



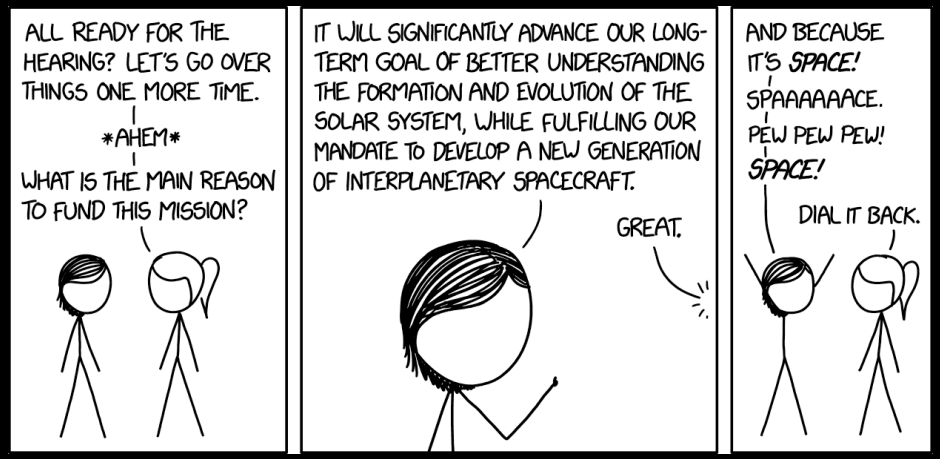
Software Updates

- Despite your best efforts some software bugs will make it to orbit
- Have a plan to address them
  - In-flight software updates
- Ensure the process works prior to launching your spacecraft



Summary

- CubeSats and software open up phenomenal opportunities in space
  - Embrace it, don't run away from it
- Include software impacts in your design-phase trade studies
- Favor terrestrial best practices over legacy aerospace practices
- Plan extra time for testing and developing testing infrastructure
- Have fun and be successful!



Questions?

- Dr. John M. Bellardo
- bellardo@calpoly.edu
- https://polysat.org