CubeSat Flight Software Development

Daniel S. Wilson

October 19, 2011

APL
The Johns Hopkins University
APPLIED PHYSICS LABORATORY

Motivation:
Typical APL NASA Mission Software Costs

- 800-1000 Staff-Months (~$20M)
- Includes:
  - Software systems engineering
  - Flight software development
  - Autonomy rule development
  - Testbed software development
  - Ground software development
  - Independent testing (IV&V)

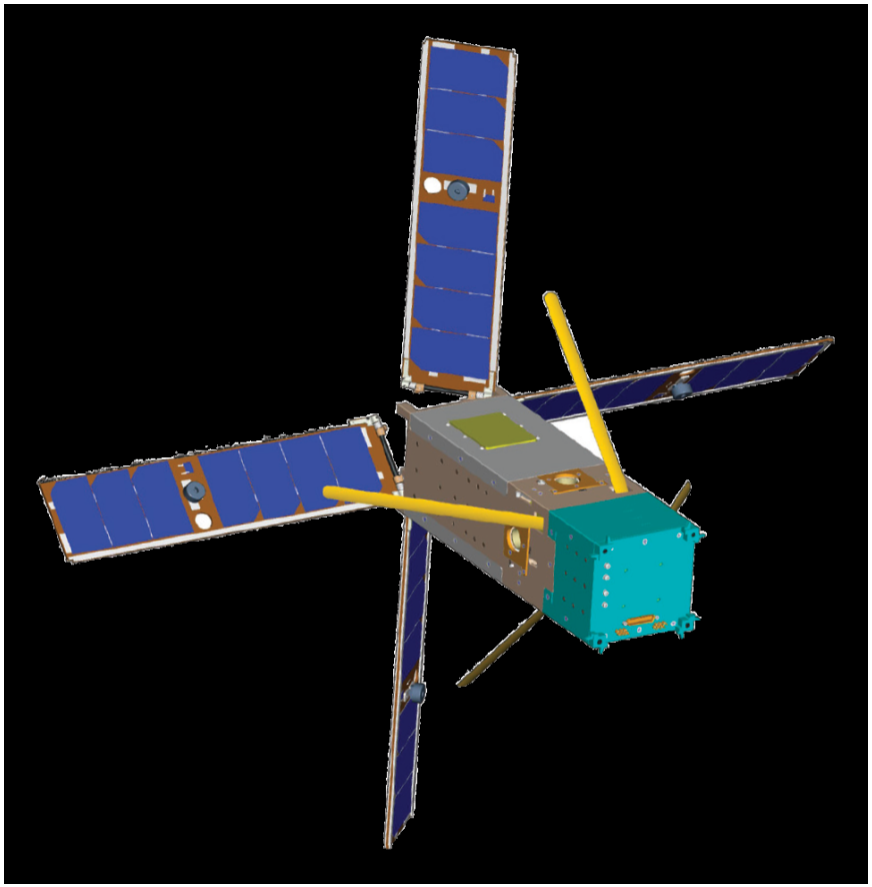
Typical CubeSat Total Mission Costs

- $100K - $6M
- Covering:
  - All hardware, software, and personnel costs

Oops!

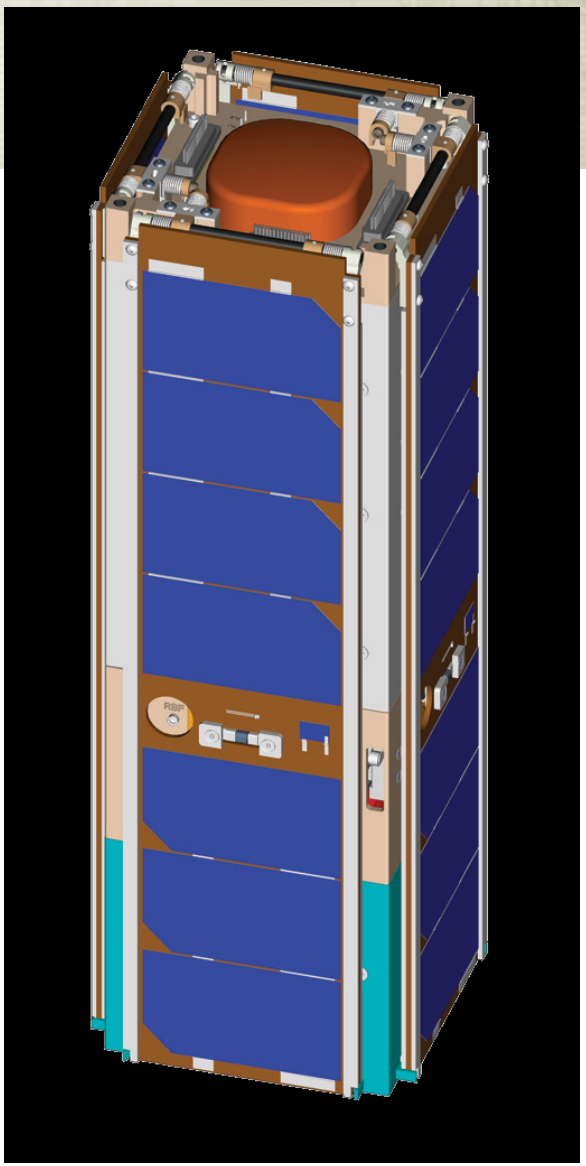
Business as usual won't work!

APL's First CubeSat



APL CubeSat Features

- "3U" form factor, meaning:
  - Volume: 10x10x30 cm
  - Mass: < 4.5 kg
- 4 double-sided solar panels that provide power in any attitude
- 3-axis stabilized attitude control
- Spacecraft processor built around a LEON3-FT CPU from Aeroflex/Gaisler
  - Radiation-hardened (fault tolerant)
  - 32-bit architecture
  - Floating point capability
- CCSDS uplink/downlink data structures used



Keys to Reducing Project Software Costs

- Reuse (starting with the concept of operations)
- Scale performance down (e.g., low telemetry rates)
- Accept more risk (e.g., no IV&V)
- Small team, short schedule
- Commercial hardware, where possible
- Skunk-works environment (co-locate, bull-pen)
- Reduced feature set, resist feature-creep
- Pay for learning curves and risk-reduction on other budgets (corporate investment in R&D)
- Management clout
- Experienced staff
- Streamlined process

APL CubeSat Software Features

- **Reused most cost-effective features from heritage systems**
  - Real-time commanding
  - Time-tagged commanding
  - Macro commanding
  - Data Summary Table for holding min's, max's, and averages of various telemetry items
- **Excluded nonessential features**
  - No autonomy engine (no redundant hardware to manage)
  - No blending of real-time and playback telemetry on downlink
  - No in-flight reprogrammability (no Boot program – huge savings!)
  - Produce much less telemetry ("learning to live with a half-duplex satellite")
  - No memory scrub (live with accumulating single-bit errors in RAM)

Hardware Characteristics that Saved Money

- Used an FPGA to provide driver-like functionality to most data interfaces
  - Spacecraft radio
  - Attitude sensors, temperature sensors, etc.
  - Attitude actuators
  - Payload electronics
- No Solid State Recorder (SSR) hardware (used an area of RAM instead)
- Single spacecraft processor (no separate G&C processor)
  - G&C task runs under the main flight application, eliminating cost of a separate app, and all its infrastructure (huge savings)

The Flip Side:
Hardware Characteristics that Increased Costs

CubeSat hardware is typically less capable due to mass/ power/volume/cost constraints. Thus, more functionality is moved to software, increasing software costs. Examples:

- Deployment sequence
- Load shedding sequence (if battery too low...)
- Half-duplex radio operations
- Heater control
- Large number of different message protocols (I^2C, NSP, SLIP, custom, big-endian, little-endian, ...)

More Cost-saving Decisions

- Open-source RTOS (RTEMS)
- Combining avionics test equipment development and flight software testbed development into a single effort
- Leveraging a Day-In-The-Life (DITL) system test to verify as many subsystem functions as possible
- Extremely simple conops for managing the telemetry data store (i.e., our virtual “SSR”)

Process Tailoring

The Software Development Plan (yes, there was one), called for reduced design reviews and no acceptance testing.

However, it called for considerable rigor during software implementation, specifically,

- **Code walkthroughs**
- **Unit testing to yield 100% path coverage**

This was intended to provide mitigation for:

- **Lack of acceptance testing**
- **Lack of ability to load code (other than RAM patches) after launch**

Reality

- Code walkthrough and unit testing discipline was relaxed after the first Build due to schedule pressure

- Result?
  - We continued code walkthroughs only on selected (~10%) new code in subsequent Builds
  - We continued unit testing on most (~75%) code
  - Quality did not appear to suffer appreciably
    - Flight software was sufficiently complete/mature in time to support spacecraft environmental testing
    - Software has proven to be very stable and robust
    - Perhaps the experience level of the team mitigated the relaxed QA

Bottom Line

- We delivered a working set of flight and ground software for ~65 SM
- Flight software source code line counts:
  - 65K SLOC (23K excluding comments) for human-generated code
  - 17K SLOC (10K excluding comments) for auto-generated code*

  82K SLOC

Compares to 300-460 KSLOC for typical APL NASA satellite flight s/w

So, by reducing the scope to ~20% of a large mission, and increasing productivity by a factor of 3-5, we reduced the overall software costs to ~6-8% of a large mission.

*The attitude determination and control software was written using Matlab© Simulink©, and then converted to C code using Matlab's Embedded Coder© tool.

Conclusions

- Software costs can be reduced for CubeSat projects, but not in proportion to the reduction in mass, volume, or hardware costs

  (one order of magnitude reduction, not two)
- Corporate investment in software risk reduction activities is necessary so that the project does not have to bear the full cost of the flight software development

  (Such investment requires clout!)
- Reuse, reuse, reuse
- Quality assurance must be reduced commensurate with cost and risk tolerance
- A repeatable process has NOT been demonstrated (we relied on heroics)
- Still need to find the right agile process
  - Can scrum work for mission-critical software?