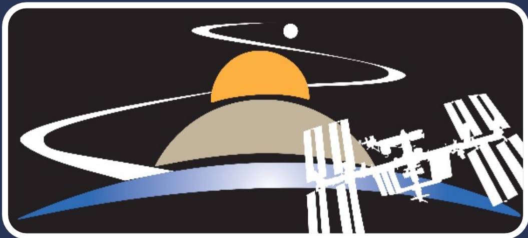
Command & Data Handling, Firmware & Software



Exploration Research and Technology Programs

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

Command & Data Handling

Approved for public release; distribution is unlimited. Public Affairs release approval
AFRL-2025-2295

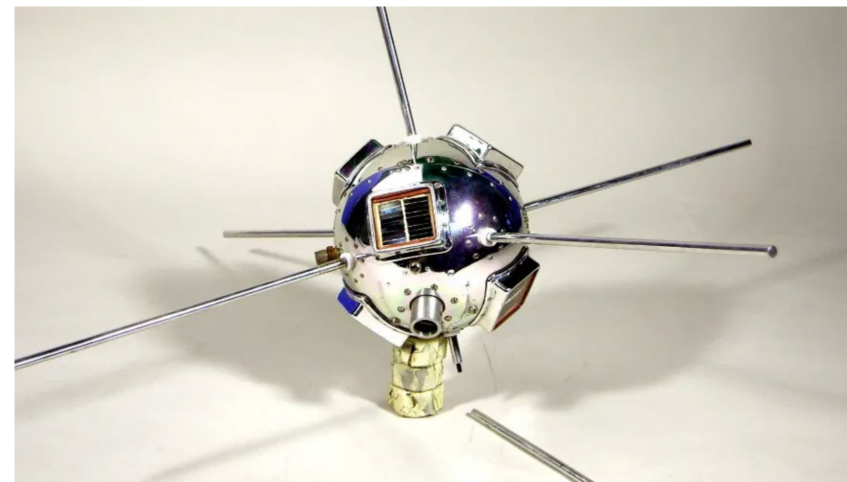
What is C&DH?

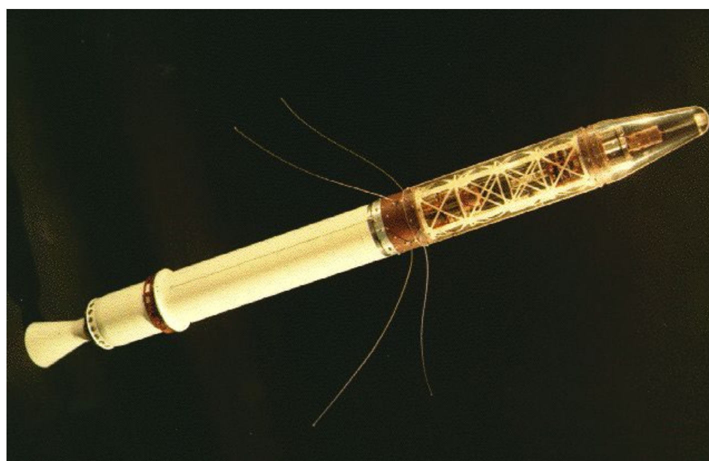
- Generates and stores all the telemetry you can; doesn't all need to be downlinked.
- Maintains team's high level telemetry to downlink during ground contacts for monitoring subsystem health.
- Has a time-to-live for state of health telemetry.
- Keeps a command requested history.
- Generates and stores subsystem metadata. Accept/reject counts, last communication time.
- Should not store telemetry from subsystems that are powered off.
- Utilizes data reduction techniques when applicable. Averaging, binning, compression
- Provides hooks for command execution on mode transitions.
- Has a way to verify command receipt with ease for in pass operations.
- Keeps a command execution history.
- Plans for the unexpected and have a native pass through command for all subsystems.
- Has commands to change telemetry generation, storage, and downlink parameters.

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

What is C&DH?

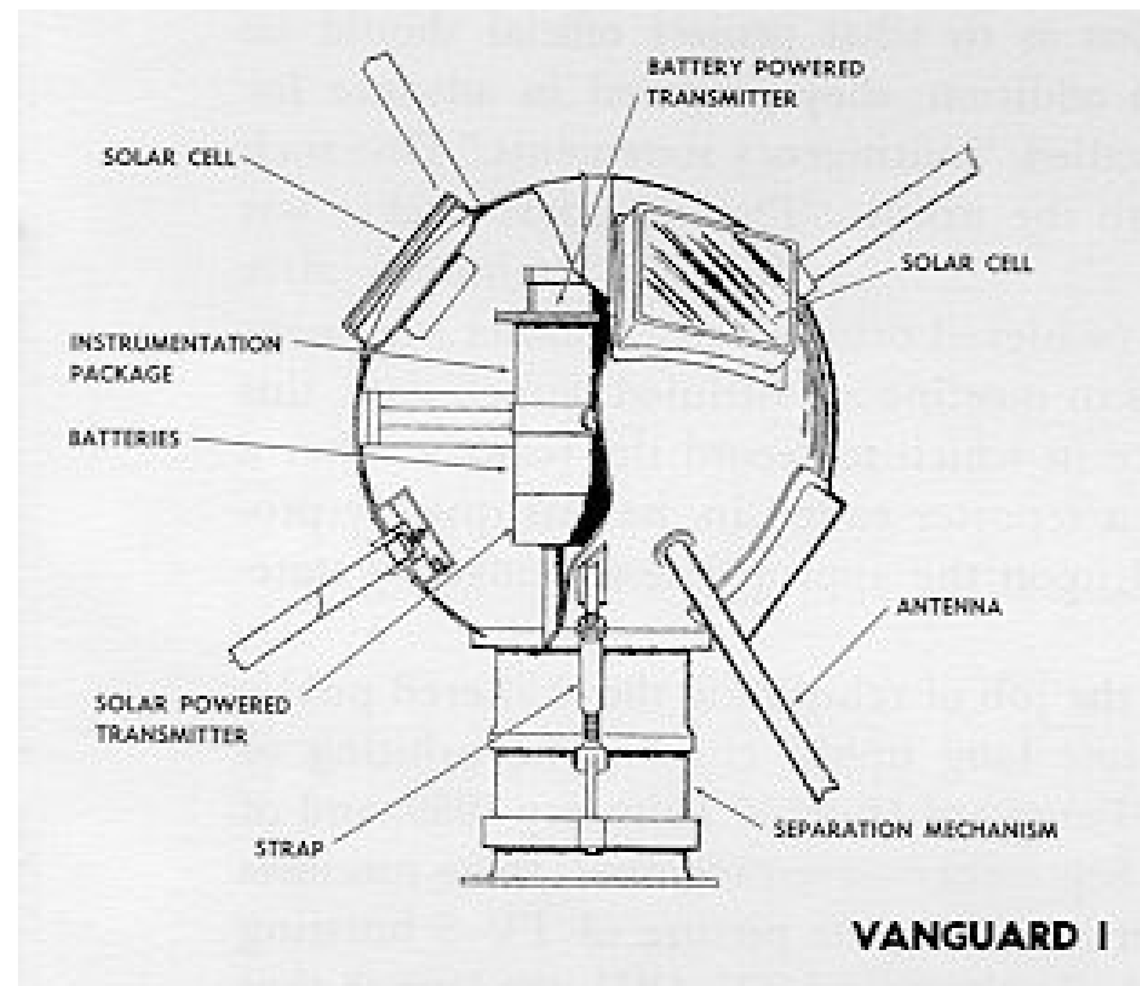
- Imagine you are operating:
  - Explorer 1: 1st satellites from USA
  - Vanguard 1: 2nd satellite from USA
- How will you talk to it?
- How does it talk to you?
- What are you saying to one another?
- How can you tell what the spacecraft is doing?





How did they do it?

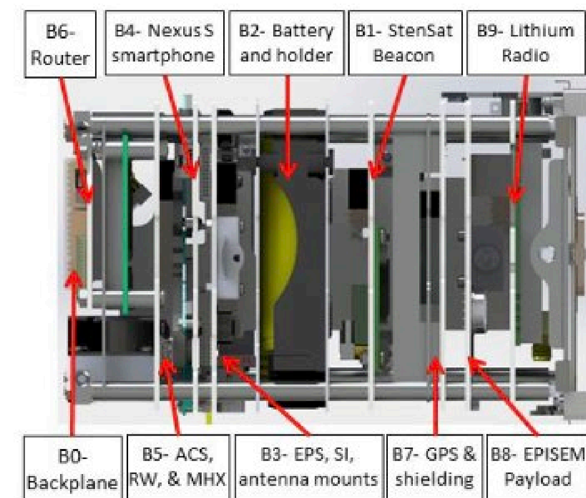
- "Direct" control via radio transmission
- Fancy signal processing to encode and decode (no computer!)
- Spacecraft always listening and becoming
- Vanguard 1 had:
  - Solar Cells
  - Battery
  - Transmitters + Antenna
  - Experiments



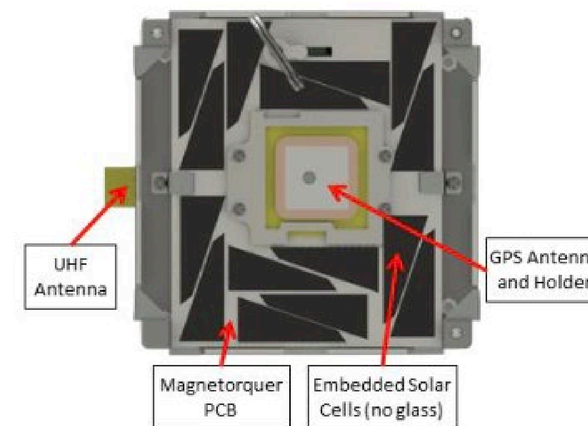
Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

How do we do this now?

- We're still doing almost the same thing
- Easier to do with modern computers
- Commercial off the shelf radios
- You're likely to have several of the same components, at minimum:
  - Solar Cells
  - Battery
  - Transmitters + Antenna
  - Experiments
  - (new!) computer
  - (new!) GPS antenna



Link to EDSN paper

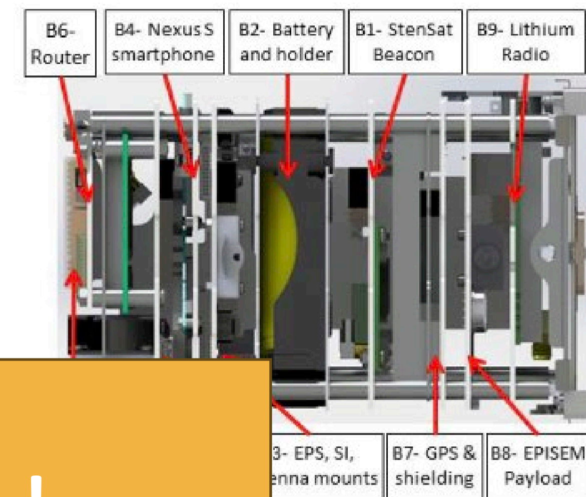


Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

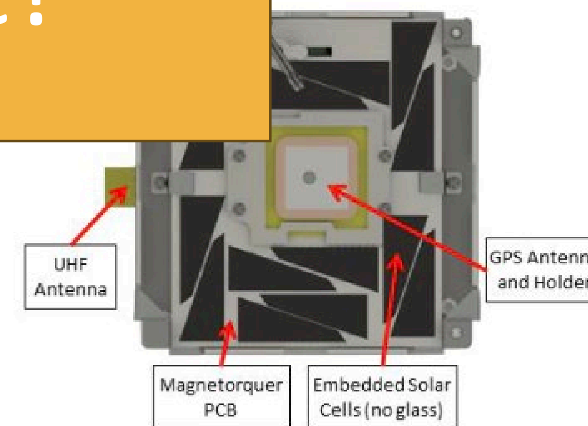
How do we do this now?

- We're still doing almost the same thing
- Easier to do with modern computers
- Commercial off the shelf radios
- You're likely to have several of the same components, at minimum
  - Solar Cells
  - Battery
  - Transmitters + Antenna
  - Experiments
  - (new!) computer
  - (new!) GPS antenna

C&DH is the glue!



Link to EDSN paper



Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

Processor Selection

1. **Bare metal or OS?**
   - Bare metal (no OS) means micro-controllers are a viable option
   - An OS on a micro-controller is possible but often not practical
2. **Power budget**
   - Instantaneous and average power draw limitations
   - Is there a sleep mode and does that help?
   - Frequency scaling using a governor may help slightly
3. **Memory (RAM)**
   - The amount of memory required to run the OS and flight software
   - Is there room to store flight software on a RAM disk?
4. **Memory (Non-volatile)**
   - The amount of memory required to store the bootloader, kernel, OS, flight software, telemetry, and duplicates for any redundant copies.
   - Can memory be added? SD card, eMMC, NAND, NOR, FRAM?
5. **Interfaces**
   - Are there enough UARTs, SPI buses, I2C buses, USB, ethernet, etc…?
   - Is the throughput high enough on each interface?
6. **Data Budget**
   - Do you have enough memory space for your experiments?
   - How often are you storing telemetry? How long can you run until you fill up your memory? How quickly can you get data down?

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

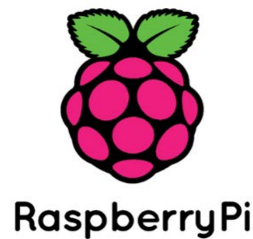
UNP Hardware

Exploration Research and Technology Programs

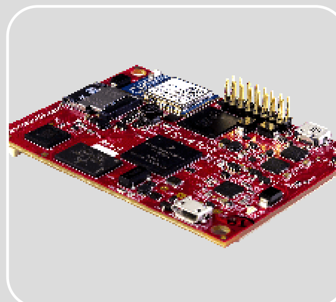
Single Board Computers

Microcontrollers

Beaglebone Black



Raspberry pi



TS-4100

MSP430FRxxx

Atmel AVR (Arduino)

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

UNP

OS Selection

OS	None (Bare Metal)	Linux	RealTime OS
Power Draw	Low (sleep modes)	Higher	Higher
Memory Consumption	Low	Higher	Medium
Scheduling	Deterministic	Less Deterministic	Deterministic
Dependencies	Few	More	More
User Base	Depends on processor	Large	Smaller
Learning Curve	Steep	Shallower	Medium

Approved for public release; distribution is unlimited. Public Affairs release approval
AFRL-2025-2295

Memory - Radiation

Problem: Memory corruption is a common occurrence in space-based applications, and steps should be taken to mitigate mission ending corruption.

- Generally orbits with high inclinations and altitudes have a harsher radiation environment.
- Memory is particularly vulnerable when being written to and energized.

Memory Types: Some memory types are more resilient to radiation. Electronic memory is the least resilient to single event upsets; SD cards in particular are known to fail on orbit. Below are some more resilient memory types to use for high importance memory.

- Phase change memory (PCM)
- Ferrite RAM (FRAM)
- Single-layer cell (SLC) NAND instead of Multi-layer cell (MLC) NAND

UNP

Memory – Radiation Mitigation

Mitigation Strategies: Shielding, error correction, and .

- Radiation hardened memory is effective, but expensive.
- Error correction
    - Error code correction (ECC), triple modular redundancy (TMR).
- Partitioning, provides some isolation to corruption and regular utilization.
- Hard reset, assuming the startup process is reliable resetting can be a viable response.

Error Correction Implementations:

MD5

- Hash comparison between recent computation and known good hash.
- No ability to correct file.

TMR

- Bit level verification and voting scheme between at least 3 file copies
- All versions scanned and majority vote wins minority bit values are corrected in differing files.

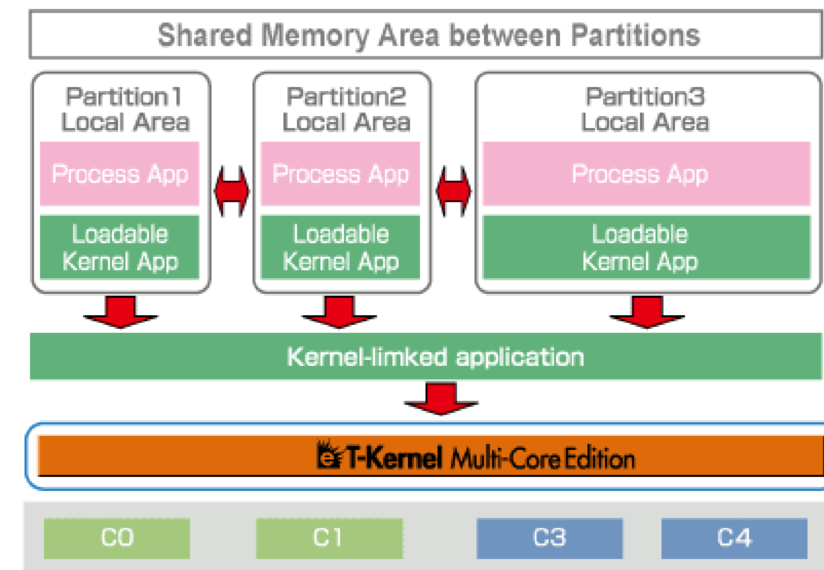
Approved for public release; distribution is unlimited. Public Affairs release approval
AFRL-2025-2295

Memory - Partitions

Partitions: Partitions help to isolate memory usage. They are helpful for preserving space in critical areas if logs or telemetry grow to unexpected size and may be used to mitigate the spreading of corruption.

Suggested Practices:

- Critical memory should rarely be written to and have its own partition.
- Low priority, high write frequency files (like logs) should have their own partition.
- Keep redundant partitions for flight software, this allows TMR across multiple partitions.
- Leave an unused partition to fail over to for things like telemetry storage.



Memory – File System

Problem: File systems that work well terrestrially are not always the best option for space based applications. The following are important factors in deciding on a file systems to use.

Memory Retirement: In flash, blocks may be marked as retired if a bad read or write occurs. Single event upsets can potentially wreak havoc on a partition by retiring large blocks of memory. Partitions may help to isolate some of this.

Power Loss: Unexpected power cycles can cause a filesystem to be corrupted. A journaling file system can be used to mitigate this, but it is still crucial to test and understand what the behavior is and potentially account for it with scrubbing or TMR.

RAM Disk: For files that do not need to persist, a RAM disk can be used instead of flash. This is particularly useful for files written to at a higher frequency because it will reduce wear and the likelihood of a single event upset.

Flash File Systems: Raw flash is handled differently than other memory types so JFFS2, YAFFS, and UBIFS are the main options.

System Startup

- Understand your process
  - Bootloader → Kernel → OS → FSW
- Identify and mitigate your risks
  - Redundant bootloader, kernel?
  - Scrub critical OS files?
  - TMR flight software?
- Handle memory failures
  - Scrub, TMR, reformat, fail over?

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

UNP Startup Stages

Hardware Stage	Software Stage	Monitor Stage

• Checkout Memory
  • Mounting
  • I/O tests
• Correct Memory
  • Disk check
  • Reformat
  • Fail over to alternate
  • RAM disk
	• Checkout Software
	  • Hash checks
	• Correct Software
	  • Scrubbing
	  • TMR
	  • Reinstall
	  • Fail over
		• Faults
		  • Trigger software restarts
		  • Trigger CDH reboot
		  • Trigger software reinstalls
		• Reverts
		  • Only if on orbit software updates

UNP Software Start-up

Flight software startup options

- Bash script to executes binaries
- Use services and Linux package management

Startup Verifications

- Startup scripts should include fail over logic in case of failure to run a binary.
- If on-orbit software updates are available, allow for reverting packages to previous version.
- Use radiation mitigation strategies

Failover Options

- A 'gold copy' of flight software should be kept in its own partition.
- OS partition backup (requires custom bootloader or other intricate design)
- Failover scenarios, such as SD -> NAND -> PCM -> RAM

UNP Serial Protocol

Asynchronous

Advantages: Polling not required
Disadvantages: Single connection
Considerations: GPIO limitations

Synchronous

Advantages: Multi-device bus
Disadvantages: Polling required
Considerations: Board layout

Single ended

- **Advantages**: Fewer data lines
- **Disadvantages**: Speed
- **Considerations**: Harness length

Differential

- **Advantages**: Speed
- **Disadvantages**: Additional data lines
- **Considerations**: Power utilization

	Asynchronous	Synchronous
Single-ended	TTL, RS-232, RS-485	SPI, I2C, OneWire
Differential	LVDS, RS-422	USB

I2C / One Wire Communication Tips

Bus capacitance

- Numerous sensors on comm bus creates additional impedance
- Mitigation 1: Adding a buffer in line can reduce
- Mitigation 2: More separate comm buses
- Mitigation 3: Digital Mux different paths to comm bus.

Bad reads

- Even healthy comms will provide bad reads occasionally.
- Add validity checking on values
- See SSC18-I-01: Dellingr: Reliability lessons learned from on-orbit

Stuck Bus lock out state

- Especially for critical subsystems
- Read more:
  - See Analog AN-686
- Mitigation 1 (I2C): Add a buffer chip like Analog LTC4308 which automatically senses a stuck bus condition and pulses the SCL line
- Mitigation 2 (I2C): Command that can send 16x clock pulses on timeout
- Mitigation 3: Fault condition that power cycles all devices on comm bus

Fault Handling: Watchdog Timer

Benefits of Watchdogs

- To detect and recover from component or communication malfunctions
- Deterministic timed response to anomalies

Possible malfunctions

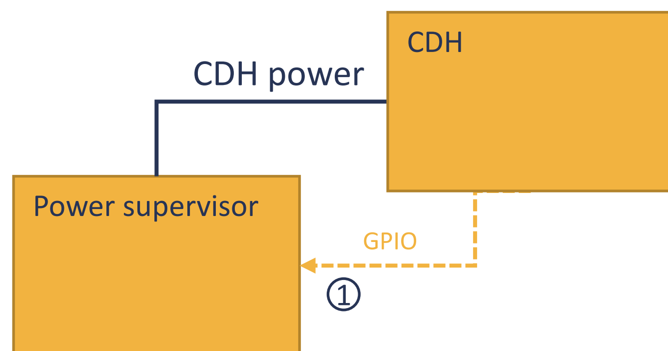
- Clock failure
- Stuck comms bus
- Program crashes

Types of Watchdogs

- Hardware-based
  - GPIO tap reset
  - Bus/component power supervisor
  - Timing often defined by RC-circuits
  - Pros: Reliable
  - Cons: Fixed
- Software-based
  - Software tap reset on Command
  - Through UART, I2C etc
  - Pros: Flexible
  - Cons: Expects higher level of vehicle functionality

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

CDH Watchdog (Simple)



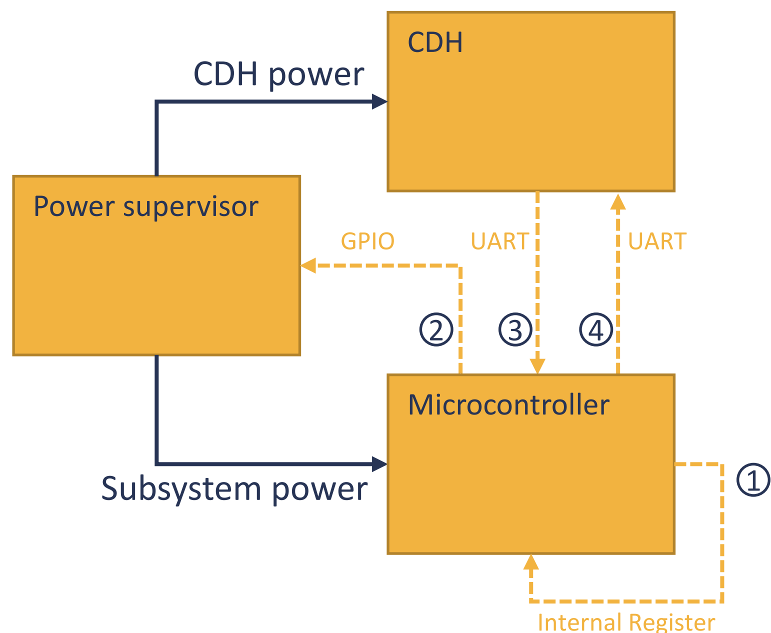
Fault – CDH fails to tap Power supervisor

Cause: FSW fault, OS fault, I/O failure, commanded reset

Effect: Supervisor removes power for 30 seconds

① CDH taps Power supervisor

CDH Watchdog (Use Case)



① Microcontroller taps itself
② Microcontroller taps Power supervisor
③ CDH taps Microcontroller
④ Microcontroller taps CDH

Fault 1 – Microcontroller fails to itself

Cause: Microcontroller fault
Effect: Microcontroller soft resets itself

Fault 2 – Microcontroller fails to tap Power supervisor

Cause: Microcontroller fault, I/O failure, commanded
Effect: Supervisor removes power for 30 seconds

Fault 3 – CDH fails to tap Microcontroller

Cause: CDH fault, I/O failure
Effect: Microcontroller starves supervisor, prompting power reset

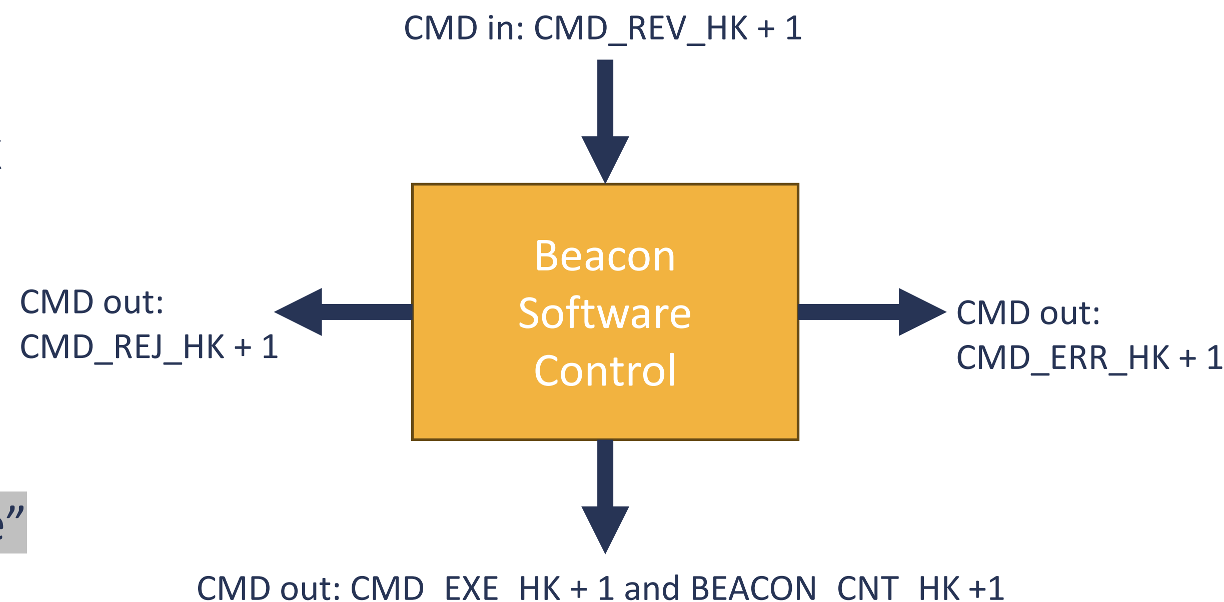
Fault 4 – Microcontroller fails to tap CDH

Cause: Microcontroller fault, I/O failure
Effect: CDH starves Microcontroller (1), (hopefully) prompting power reset

UNP Housekeeping and Counters

- The bread and butter of simple telemetry
- Like watchdogs, but between/within software components
- The more the better (mostly)
- Standardize HK conventions
  - When do you reset them?
  - Do you let them roll over?
  - What's the largest number you want store?
- Interdependent HK relationships give great insights
  - CMD_REV_HK = CMD_EXE_HK + CMD_ERR_HK + CMD_REJ_HK
- Simple Counters are great too
  - BEACON_CNT_HK++
- Your system should have HK packets
  - Contain your HK counters
  - Verbosity should be adjustable by system
- The smart way to do it: instead of just doing print "I got here"

HK Type	Count
CMD_REV_HK	100
CMD_EXE_HK	96
CMD_ERR_HK	1
CMD_REJ_HK	3
BEACON_CNT_HK	96



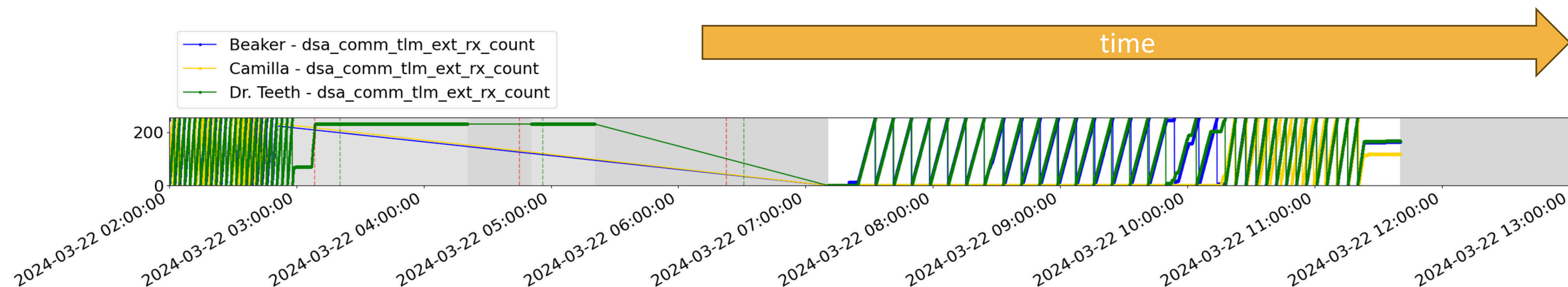
UNP Housekeeping Packets

- The bread and butter of simple telemetry
- To be viewed over time
- Structure dependent on software design
  - Per software application or
  - Basic system HK always enabled or
  - HK is always on

ID	Value
Time	100000
APP ID	0x01
CMD_REV_HK	100
CMD_EXE_HK	96
CMD_ERR_HK	1
CMD_REJ_HK	3
BEACON_CNT_HK	96

ID	Value
Time	100001
APP ID	0x01
CMD_REV_HK	101
CMD_EXE_HK	97
CMD_ERR_HK	1
CMD_REJ_HK	3
BEACON_CNT_HK	97

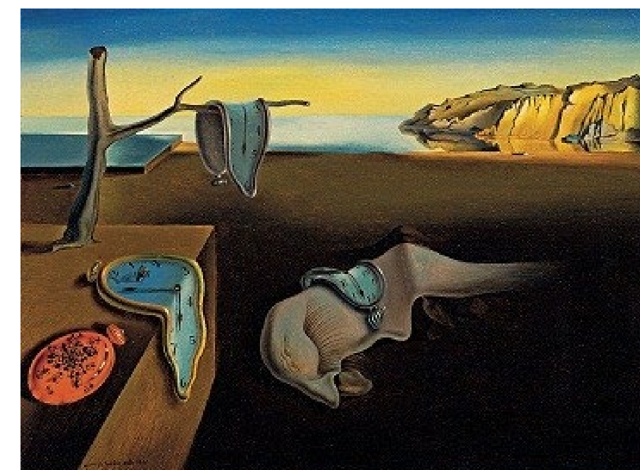
ID	Value
Time	100002
APP ID	0x01
CMD_REV_HK	102
CMD_EXE_HK	97
CMD_ERR_HK	2
CMD_REJ_HK	3
BEACON_CNT_HK	97



UNP

Synchronizing and Managing Time

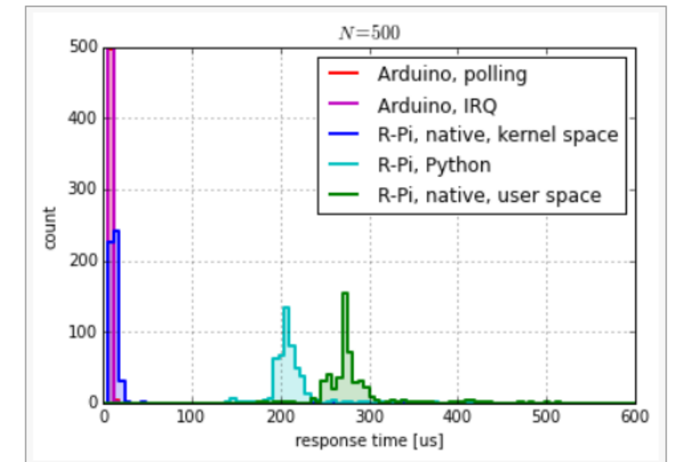
- Time requirements affect many subsystems and satellite operations
  - Command execution accuracy
  - Telemetry timestamp accuracy
  - ADCS pointing accuracy based on knowledge of position, velocity, and time
  - Synchronizing telemetry collects for payload(s)
- Time management on board a satellite needs to handle varying time knowledge circumstances to account for various CONOPS, fault handling, and GPSR lock conditions
  - Satellite reboots
  - Powering off GPSR due to entering low power modes
  - Often a GPSR will be restarted as a fault response
  - Attitude maneuvers may cause degradation or loss of GPS time knowledge
- Time sources, standards, and leap seconds may need to be accounted for
  - UTC (GMT): -18s from GPS *(currently)* , -37s from TAI *(currently)*
  - GPS: +18s from UTC *(currently)*, -19s from GPS *(always)*
  - TAI: +19s from GPS *(always)*, +37s from UTC *(currently)*
  - Linux epoch: Jan 1st 1970 00:00 UTC
  - GPS epoch: Jan 6th 1980 00:00 UTC



local	2022-04-06 10:15:26	Wednesday	day 096	timezone UTC-6
UTC	2022-04-06 16:15:26	Wednesday	day 096	MJD 59675.67738
GPS	2022-04-06 16:15:44	week 2204	317744 s	cycle 2 week 0156 day 3
Loran	2022-04-06 16:15:53	GRI 9940	380 s until	next TOC 16:21:46 UTC
TAI	2022-04-06 16:16:03	Wednesday	day 096	10 + 27 leap seconds = 37

Synchronizing and Managing Time

- Synchronizing to GPSR time with PPS
  - GPS position, velocity, and time are accurate at a given PPS edge
  - GPS time messages are sent after the PPS edge they correlate with
  - GPS disciplined services may exist could be used with your GPS (e.g. chrony)
- Hardware and firmware/software implementation drives accuracy of time knowledge when relying on PPS
  - Linux OSs are not real-time and have more limitations in PPS based time synchronization
    - Interrupt driven events can have large latencies and jitter in some implementations
    - Linux user space and kernel space implementations vary drastically
  - FPGAs and microcontrollers are much more accurate in responding to GPIO interrupts like PPS
- Synchronizing time-based between CDH and data collects
  - Starting collects on PPS edges can be a vary accurate way to start a data collect. This method can reduce the timestamping accuracy requirement.
  - If disturbing the PPS signal to multiple subsystems, a signal buffer may be needed
- Maintaining time without GPS and across reboots
  - Large jumps in time should be avoided, so time should be propagated accurately in cases of loss GPSR time
  - RTCs can be used to keep moving time forward which is important for telemetry timestamping



Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

UNP Testing Interfaces

Exploration Research and Technology Programs

Integrated Functional Testing Considerations

Radiating over an RF path isn’t always allowed or possible.

- Redirect flight software to a test interface and FEP
- Use an RF switch and attenuators to directly connect radio to ground station
- Turn file logging up and use ssh to verify functionality.

During thermal vacuum testing, the satellite will be powered down during temperature transitions.

- Provide hooks to EGSE to read temperature sensors

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

Firmware & Software

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

UNP

Firmware vs Software?

	Purpose	Design and Structure	Execution	Size and Complexity	Updates and Maintenance
Firmware	Control specific hardware device or system. A low level interface controlling memory, I/O operations, and peripherals.	Low-Level programming languages: assembly or machine code. Handles things like hardware interrupts and	Always active, even when some software is not running	Small, simple, singular tasks with limited functions related to hardware control. Complexity becomes static eventually	Not updated often. Updates may require special tools for flashing/updating chipsets/ICs.
Software	Running tasks or applications. Utilizes underlying hardware functions to	Written in high-level programming languages: C/C++, Python, Java, … etc;	Runs on top of firmware, uses firmware to access hardware; is updated, installed/uninstalled independent of firmware	Can be complex and large, even millions of lines of code. Complexity grows over time.	Released frequently, updates independently of hardware,

Firmware vs Software

- Was that just jargon?
  - Sort of… but it helps to think of:
- When designing subsystems and components
- When interfacing with peripherals
- When using a combination of IC, Microcontrollers, and SBCs
- When considering the anticipated frequency of functional updates
  - How often do you expect this component to need updates?
- If you are designing and building a PCB, you will likely develop firmware first
  - This of the c standard library: what types of support does it give?
    - Example: timing, memory I/O, thread management
- What needs HK counters vs Watchdogs?

You Need a Software Workflow

- Define: How and when will things occur
  - Formal software reviews
  - Peer reviews
- Define: How are priorities decided
- Define: How is tasking assigned
  - Plan out tasks in advance and track progress
  - Use trello, bitbucket tasks, Jira, etc…
- Have frequent tag-up meetings
  - Share progress
  - Ask for help, talk about blockers
- Continuous integration
  - Doxygen
  - Style check
  - Valgrind

UNP Organization Method – Waterfall Methodology

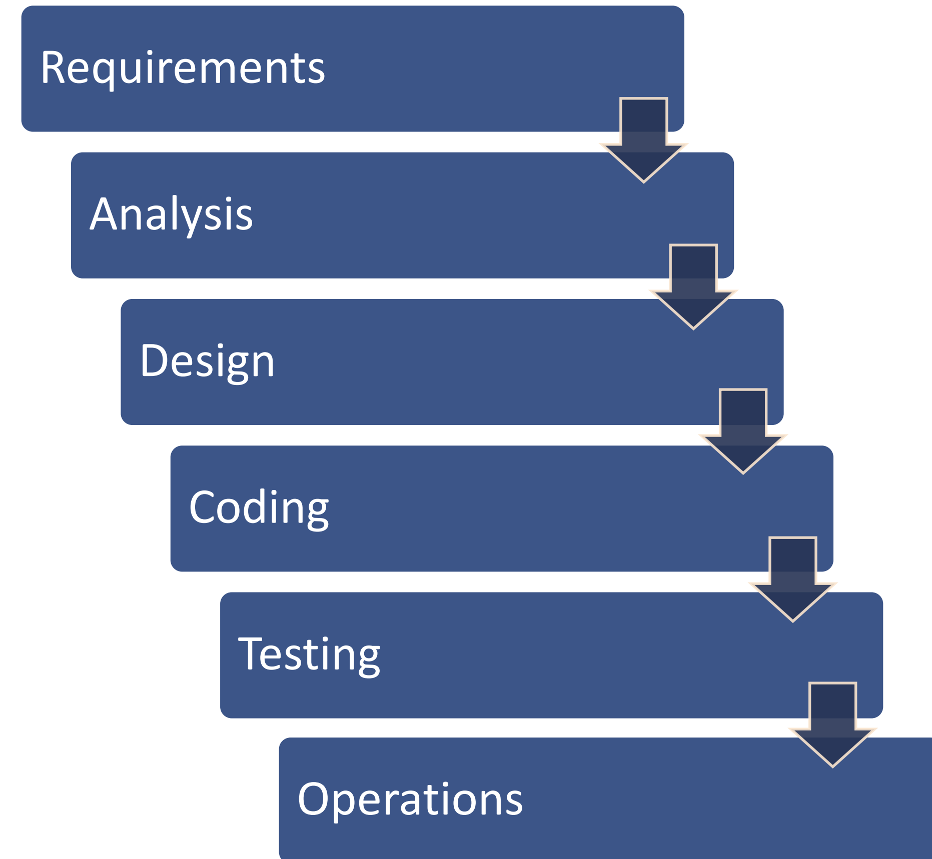
- Originates with manufacturing and construction processes
- Project mindset
- Slow phase transitions
  - Each phase depends on the deliverables of the previous ones

Pros

- Clearly links requirements to production
- Useful tool for defining milestones

Cons

- “make it up before you start, live with the consequences”
- Deadline creep – every phase delay delays the next



en.wikipedia.org/wiki/Waterfall_model#Model

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

UNP

Organization Method – Agile Methodology

Exploration Research and Technology Programs

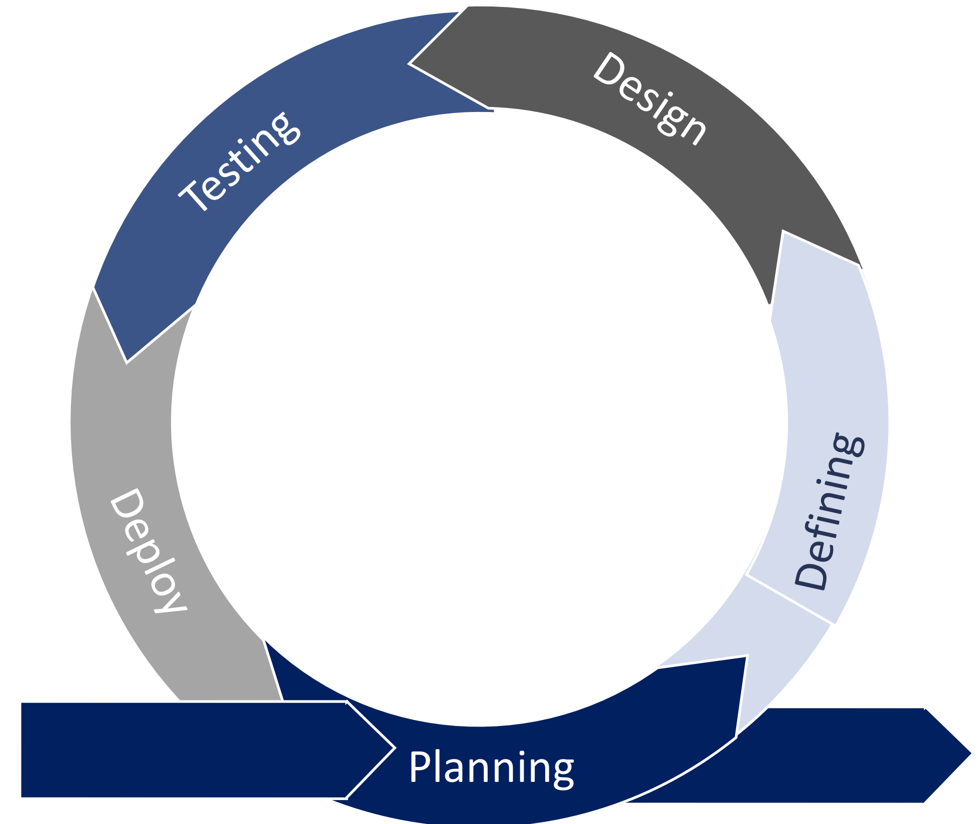
- Originates with software development
- Adaptive mindset
- Frequent phase transitions
- Work divided into iterative cycles
  - Ex. Time constrained, or feature constrained

Pros

- Flexibility for scheduling work
  - Ability to re-vector based on priority (hardware availability, etc.)
- Ability to rapidly <u>correct</u> issues
- Providing releases early (Minimum Viable Product)

Cons

- "make it up as you go along"
- Ability to rapidly <u>create</u> issues
- Does not inherently guarantee requirements are met

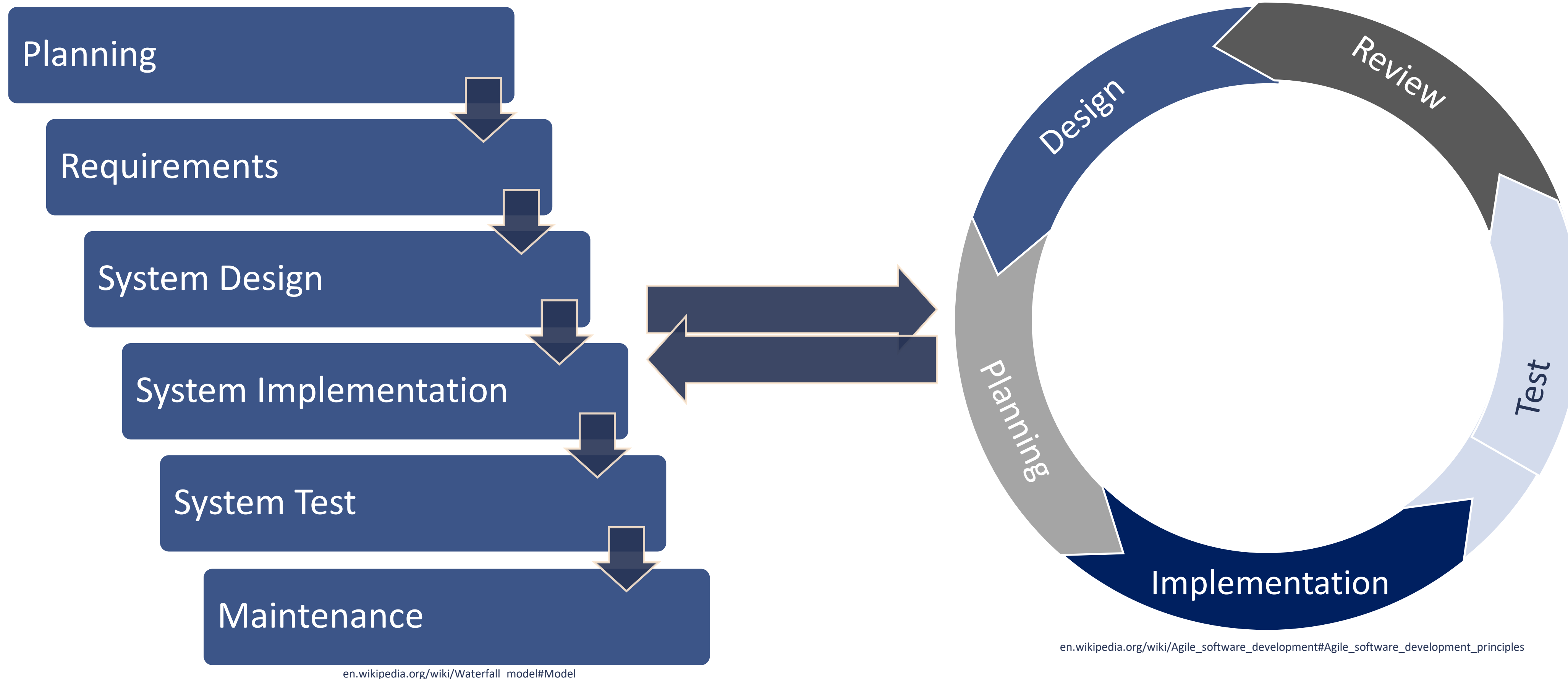


en.wikipedia.org/wiki/Agile_software_development#Agile_software_development_principles

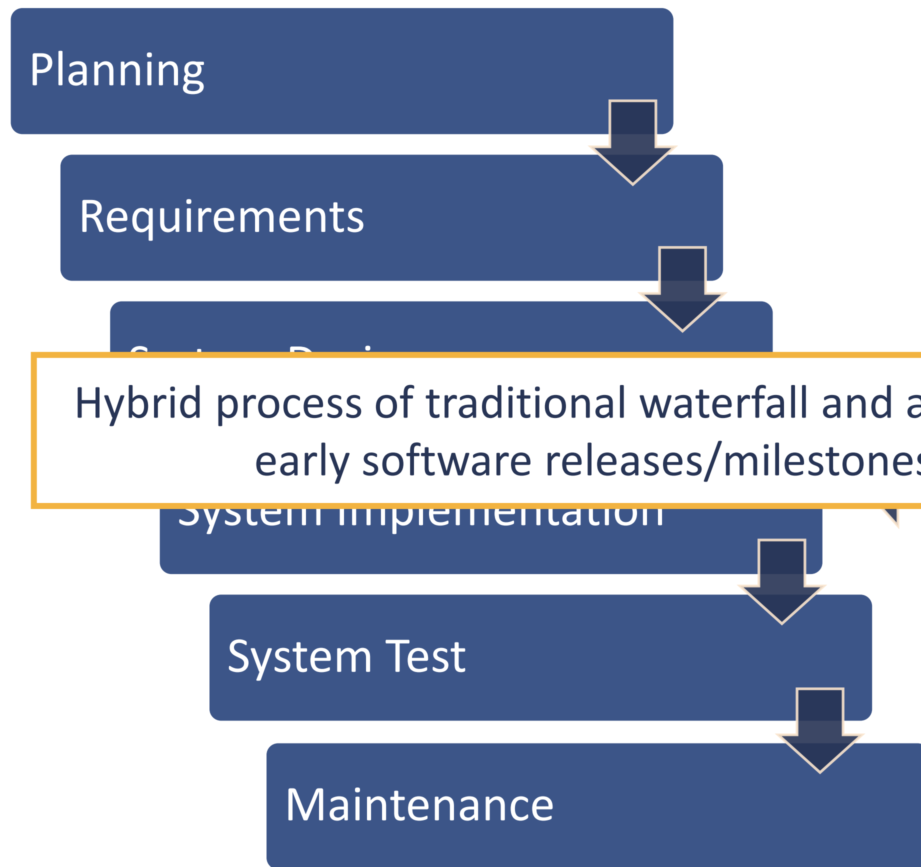
Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

Organization Method – AgileFalling Methodology

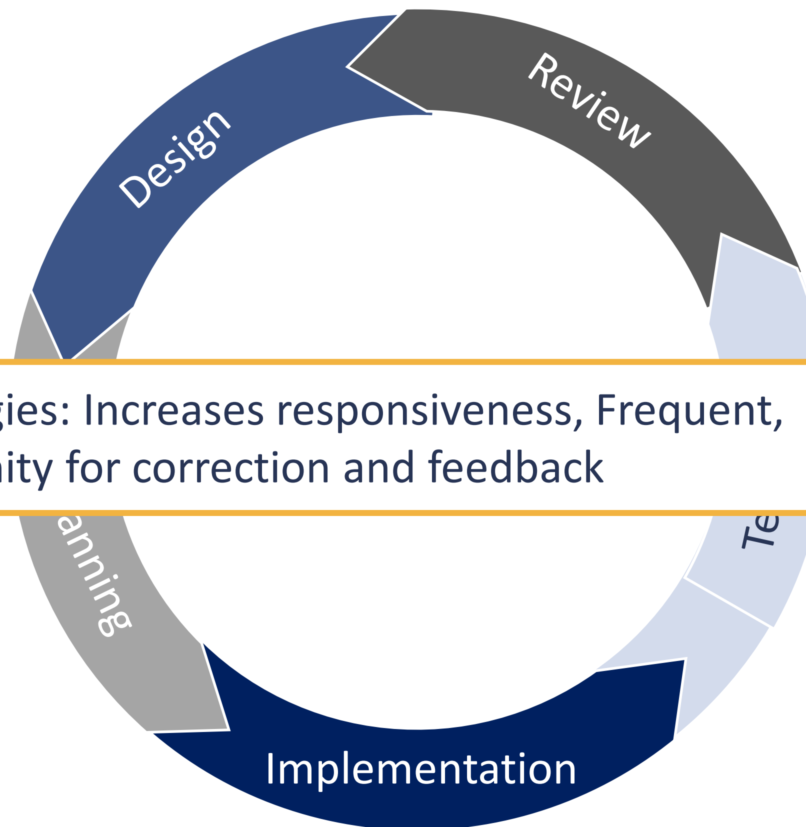
Exploration Research and Technology Programs



Organization Method – AgileFalling Methodology



en.wikipedia.org/wiki/Waterfall_model#Model



en.wikipedia.org/wiki/Agile_software_development#Agile_software_development_principles

Hybrid process of traditional waterfall and agile methodologies: Increases responsiveness, Frequent, early software releases/milestones, Early opportunity for correction and feedback

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

Version Control Systems (VCS)

- Version Control Systems is the practice of tracking and managing changes to digital files
- Keeps a complete history of file changes
  - Tracking every modification in a special database
  - Allows roll-backs and version comparisons
- Allows code branching for parallel feature development
- Using a repository provides a home for code that accessible by entire team
  - Is not just one programmer's laptop
- Complements the team's workflow with deliverables and testing support

Version Control System Tools
Git
Mercurial
SVN (Subversion)

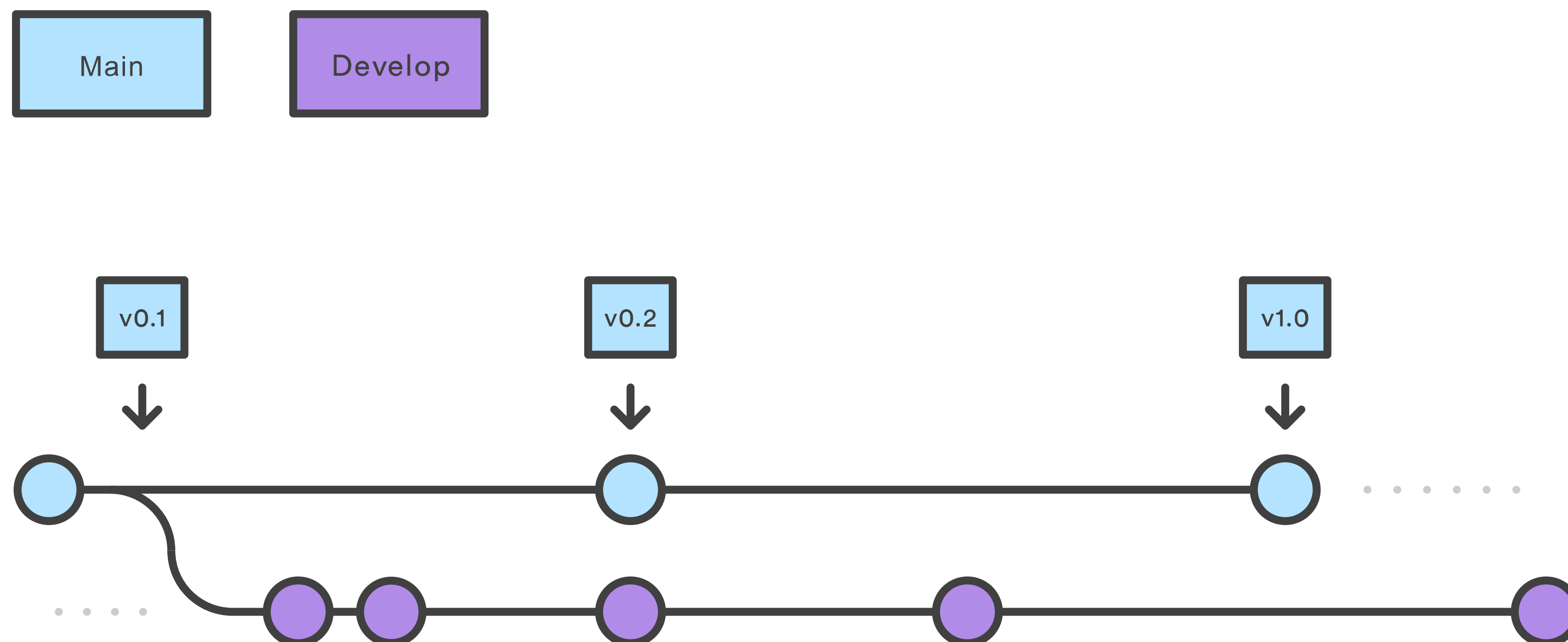
Requirements

- Driven by mission objectives: Define your Minimum Viable Product (MVP)
- Hybrid of traditional requirements used to derive user stories
- User stories: try to stay away from how and focus on what and why
  - WHO “wants to” DO A SOMETHING “so that” THERE IS SOME BENEFIT/OUTCOME
  - Define “Definition of Done” – testable outcome/product
  - Make sure to time box, e.g. if you cannot finish the story in one sprint then it should probably be broken up (could be steps in a larger process)
  - Incorporate feedback
  - Benefits
    - Keep focus on user/operator/consumer
    - Drive creative solutions
    - Enable collaboration
- Other user story considerations
  - Epics – groups of user stories and associated features to realize a larger goal
  - Spikes – information gathering or decision needed to move forward on another story or feature

Approved for public release; distribution is unlimited. Public Affairs release approval
AFRL-2025-2295

Git Workflow Example

- Main Branch
  - You don't touch it!
- Develop
  - Where your team puts their work after they have completed it
- Each dot here is a "commit"
  - When you push your code up to the main version control system
  - When you commit you have saved your progress and backed it up
  - Others can "pull" your commit and see it



Git Workflow Example

- Feature Branches
  - Where you do your actual work
  - Asynchronous work occurs here
  - The software development process consists of:
    - Creating feature branches
    - Writing code in feature branches
    - Testing feature branches
    - Reviewing other's feature branches

Main | Develop | Feature | Feature

v0.1 | v0.2 | v1.0

UNP Git Workflow Example

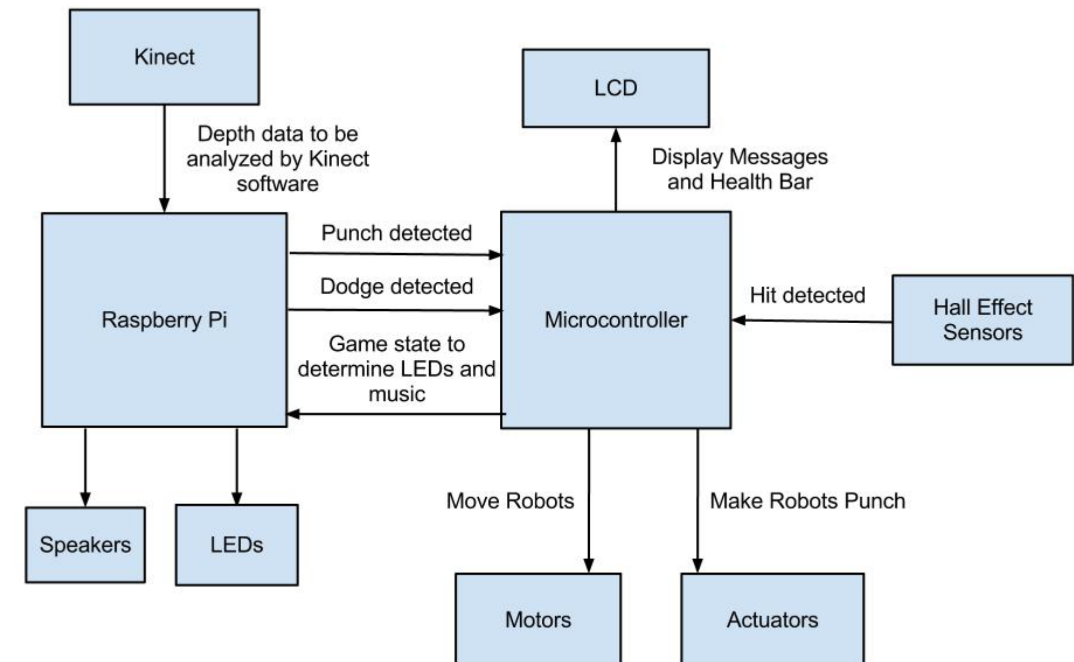
- Releases are how you safely merge
  - Develop into Main
- Releases occur:
  - Typically, every 2-8 months
  - At the end of long build cycles
  - Before major reviews
  - During Flight and Operations
    - These should always be backwards compatible
- This merge will consist of verification and testing with all the testing you have
  - PiL or DiTL
  - Unit Testing
  - Integration testing

Main | Release | Develop | Feature | Feature

v0.1 | v0.2 | v1.0

Design: Diagrams

- Software Block Diagrams
  - Capture process and data flow
  - Interaction between components (application or modules)
  - Define interface between blocks (parallel development, robust)
  - Expand blocks to design/develop functionality
- Sequence diagrams
  - Processing/handling constraints
  - Illustrates where things can go wrong and how issue will be handled
- Design Concepts
  - Modularity – promotes reuse and parallel dev
  - Low coupling
  - Hardware abstraction – reusability and testing
  - Simplicity – ease of use and maintainability
  - Target configuration versus hardcoded changes
  - Timing considerations ☺ (synchronization, persistence)
  - Define faults and fault handling early
- Timing Diagrams
  - Processing time, Timeouts, Timing Dependencies



Design: Common Code

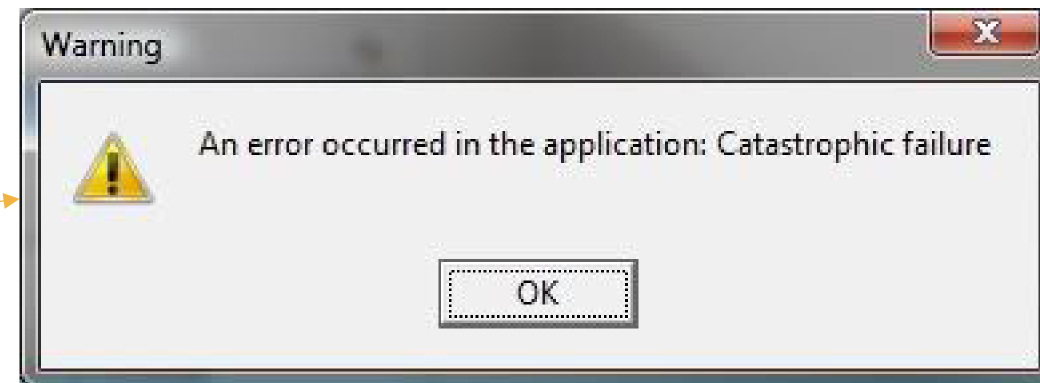
- Develop generic reusable code, modules, plugins
  - Depending on your library how you implement this may vary
  - BufferUtils
  - Common housekeeping, watchdog handling
- Utilize configuration files
  - Have a common style: Xml, environment files, command line arguments
  - What strategy you use depends on how you want to make changes on orbit
- Shared memory
  - Common values can be shared time, reboot count, etc.
  - Synchronize timing

Approved for public release; distribution is unlimited. Public Affairs release approval
AFRL-2025-2295

Design: Faults

- Basic concept: detect a condition and respond
- Responses:
  - Hard & soft resets
  - Change mode
  - Restart application
  - Log an error
  - Other
- Common Faults:
  - Last successful uplink (days)
  - Time since comms with subsystem
  - Low battery
  - ADCS momentum too high
- Watchdogs

Bad design

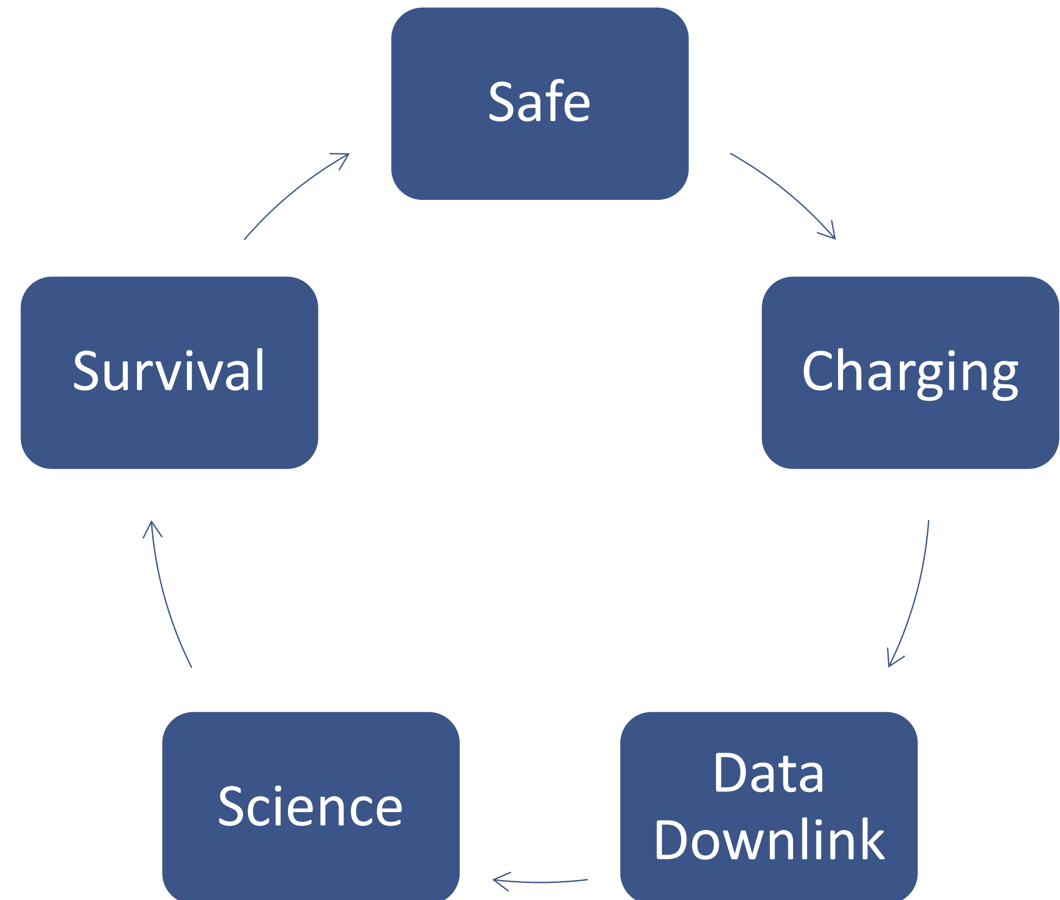


good design

```
[debug:00001] Step 1 successful
[debug:00002] Step 2 successful
[error:00003] Step 3 failed in function DoTheThing() with return code 42
[warning:00004] Step 4 was skipped
```

Design: Modes

- Modes should be known good states
  - Useful for recovery from fault conditions, set everything to known states
- Things to define
  - Telemetry gathering
  - Subsystem power states
  - Fault conditions



UNP Implementation

- Document as you go
  - Language supported or independent (Doxygen, xml)
- Follow Language Coding Standards (C/C++ as an example)
  - Easier to review/understand, easier to maintain, more consistent
  - References
    - C - https://users.ece.cmu.edu/~eno/coding/CCodingStandard.html
    - C++
      - Hits the high points: https://users.ece.cmu.edu/~eno/coding/CppCodingStandard.html
      - Comprehensive: http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
        - More than just style
  - Enforcement
    - Linters, clang-tidy
- Issue tracking (use tools)
  - Capture notes, related resources, and progress (continuity between developers)
  - Relay stage/status (ticket workflow, e.g. in Design, Implementation, Review, Testing)
  - Tie issues/features to SCM system (e.g. Jira-> Bitbucket)

UNP Review Structure

Exploration Research and Technology Programs

Peer Review

- FSW code review process followed with minimum of 2 reviewers
- Benefits
  - More in-depth review than is possible in a sit-down meeting (e.g. component ratings)
  - Tight collaboration within the team to keep all pieces progressing on a consistent design
  - Catch bugs early
  - Improved team awareness of code

Testing – Unit Testing

- Invest in Continuous Integration testing tool
  - Validates all of the unit tests
  - Compile source and documentation
  - Static analysis (cppcheck, clang-tidy, cpplint, vera++, CHAP, LLVM/Clang Sanitizers)
  - Dynamic analysis (valgrind and subtools, LDRA)
  - Generate reports
  - Know what broke and when
  - Example Tools: Jenkins, Travis CI, Bamboo, GitLab CI
- Tie-in for testing/test results against user stores/requirements (e.g. Jira Test tickets linked to User Story tickets)
- Develop simulators/emulators to support testing
  - Develop them only to the level needed for testing
  - Use mocks/stubs when possible (easily control inputs to interface)
    - Target abstracted hardware interfaces
- Feature/Unit Testing
  - Define coverage goals
  - Write meaningful tests
  - Leverage existing frameworks, e.g. gtest, nunit
  - Confirm core functionality

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

Testing – System/ Integration Testing

- Exercise each piece thoroughly and then incrementally build up (easier to track what caused an issue)
  - Confirm the full system interoperates as expected
- Define your test flow
  - How you build up your testing and system
  - Gates that must be passed to move to next phase
- Test Hardware-In-The-Loop (HIL) ASAP
  - flat-sat, segmented test hardware
- Typical types/categories:
  - Acceptance Testing Procedure (ATP) - releases/deliveries
  - Abbreviated Functional Tests (AFT) - minor/limited updates, hardware verification after environmental testing
  - Full Functional Tests (FFT) - hardware checkouts, larger code updates, hardware verification after environmental testing
  - Day In The Life (DiTL)
    - Target as much real hardware as possible in ops environment
    - Verify system meets requirements/mission objectives
    - Train "operators"
    - Find pain points, learn how to resolve anomalies on the ground
    - Tune configuration
  - Fault testing
    - Test against hardware (when safe and possible)
    - Use simulators remaining for any faults
- Automated where/when possible (trade between time to automate versus repetition)
  - Reduce error
  - Automatically capture results
  - Test early and often
  - Test-as-you-fly – ideally automated test environment, scripts, tools, etc can be leveraged for operations

Approved for public release; distribution is unlimited. Public Affairs release approval
AFRL-2025-2295

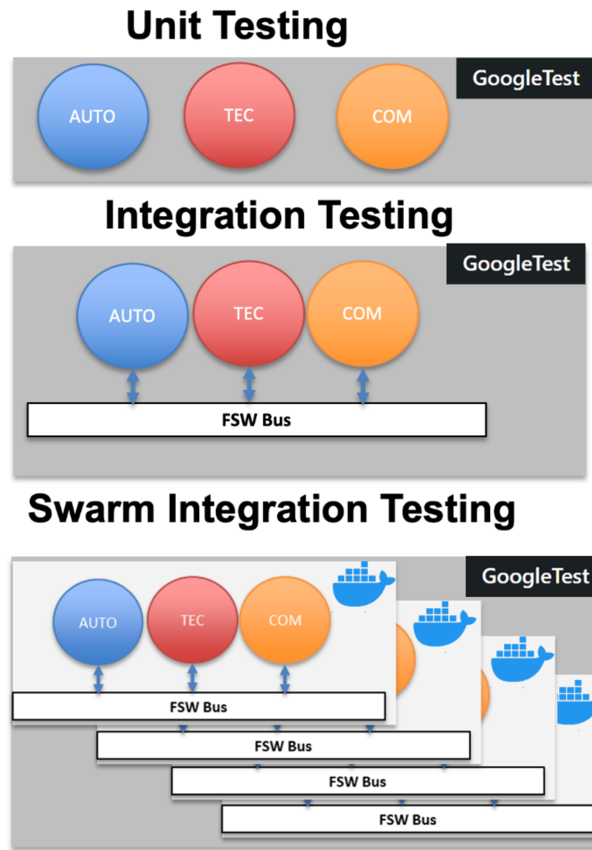
Testing Coverage and Testing Philosophy

- You should have more test code than flight code
  - Every time you write flight code it should have an associated test
- 1 unit test per fsw function gives 100% converge
- Don't cut corners – it's better to actually have missing coverage than it is to have "fake" coverage
  - Example: padding a packet with all 00000s instead of using the real format
- Skipping redundant tests at higher lower levels makes it harder to narrow down a problem
  - "We already have an integration test that covers this"
- Every line of test code you write helps you find a future problem faster
- Test code is also a FSW tutorial and documentation

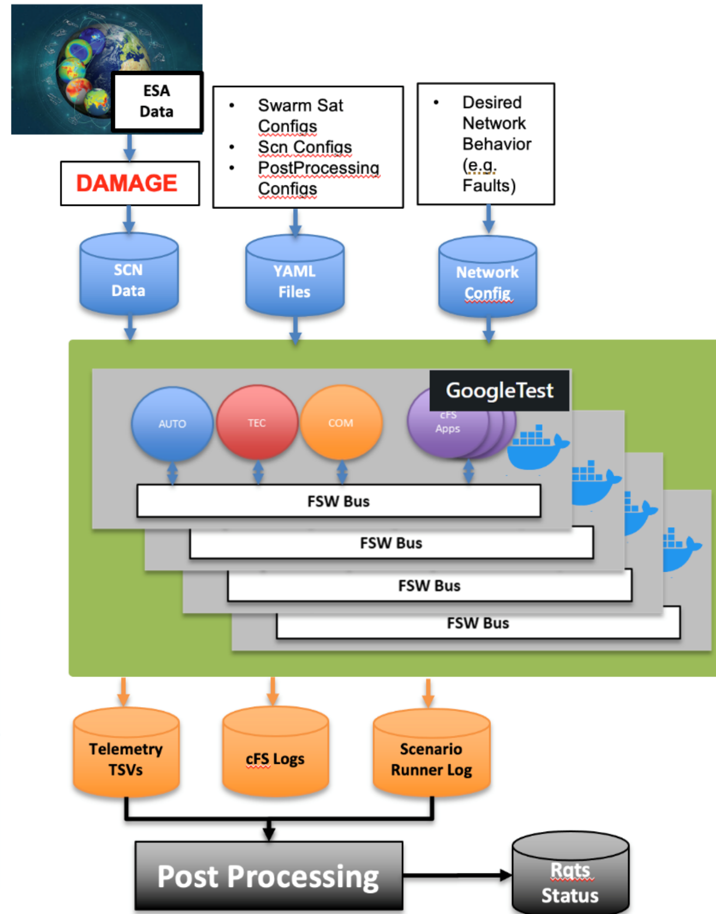
Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

Example Test Framework

Test Types

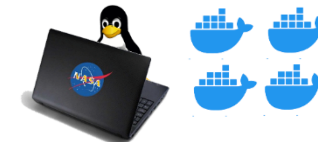


DSA Scenario Testing

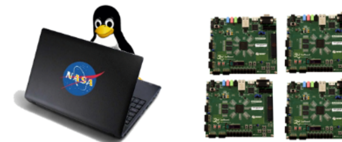


Test Environments

Containerized Testing Environment



DSA PIL Testing Environment



Starling HIL Testing Environment

Flight Processor/HW Fidelity

*test framework from DSA testing system

UNP Maintenance

- Types
  - Corrective – fix bugs
  - Adaptive – account for hardware changes/degradation or new stretch goals
  - Perfective – address things like ease of operations
  - Preventative – forward looking to avoid serious problems in the future
- Mission lifespan support
  - On-orbit updates: functional changes, bug fixes
    - Define release process, testing process, update process (upload procedure and verification)
  - Plans for handling degradation of hardware: solar array efficiency, battery capacity
    - Ideally configuration changes
- Reuse in other missions/programs
  - Plan for LTS or ROI
  - Bug fixes/tracking
  - Improved design
  - Implement enhancements
- Documentation
- Retire/EOL – source, build artifact preservation

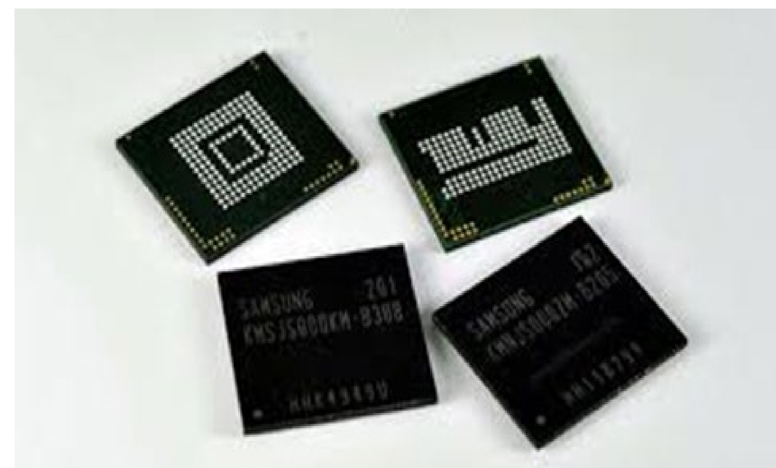
Approved for public release; distribution is unlimited. Public Affairs release approval
AFRL-2025-2295

Data Budget

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

UNP Data Generation

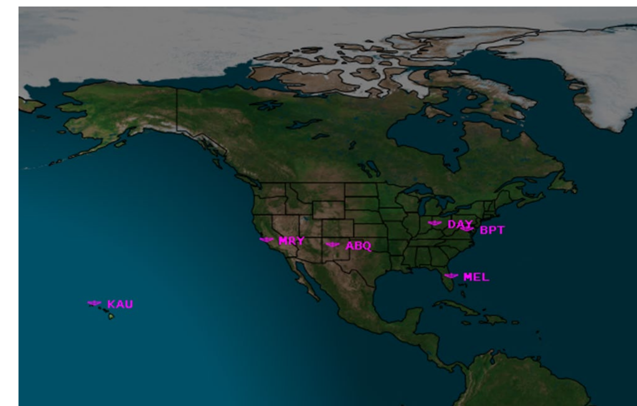
- Determine data sources
  - Payload/experiment, Spacecraft, SOH, Commanding
- Determine data rate, overhead, duration, and frequency
  - May vary depending on spacecraft mode
    - Separation, sun safe, standby, data collection, communication, etc.
- Determine store and forward requirements
  - Volatile & Non-volatile (persistent across power cycles and reboots) memory
- Consider Margins of Error
  - What if you miss a ground contact?
  - What happens when your data fills?
    - Does it overwrite something important for the OS?



UNP Communications

Typical data budget dictated by ground system resources

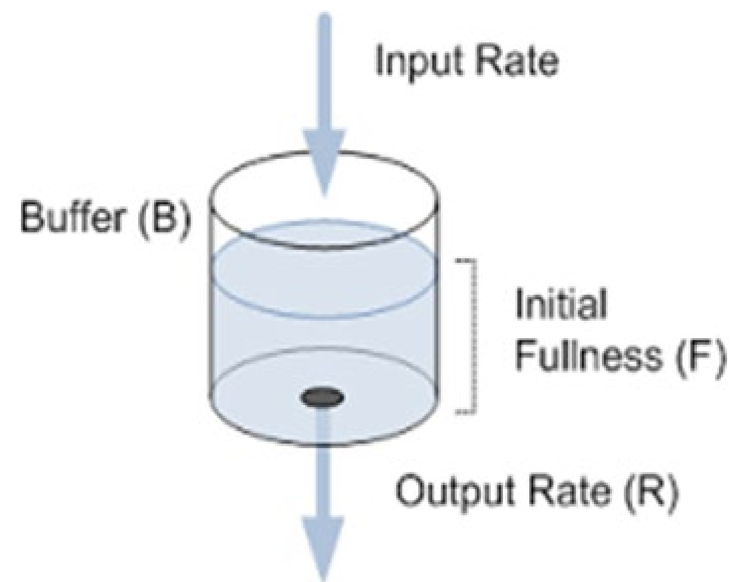
1. Identify available ground systems and contact times
2. Identify approximate communication rate
   - Uplink and downlink rates will most likely be different (e.g. UHF vs S band)
   - Uplink/downlink may be half or full duplex
3. Data budget:
   - bits/contact = Link time (sec) * link rate (bit/sec)



Data Volume

What if data generation exceeds data downlink rate?

- The obvious – generate less data
  - But… can mission objectives still be met?
- System engineering
  - More capable radio and/or ground stations
  - Add ground stations
- Data compression
  - Lossless - all data recovered when data is uncompressed
    - Generally required for engineering data values
  - Lossy
    - Generally used for image data where image degradation is acceptable
  - Processing resource considerations
    - How will data be compressed?
      - Specialized hardware e.g. FPGA
      - Satellite avionics single board computer



Memory Management

What if Satellite is unable to communicate with ground system(s)?

- Possible causes - ground system maintenance, scheduling conflicts with other satellite(s), signal strength/noise, etc.

- Need to determine how to mange data accumulation
  - If not managed satellite could eventually run out of memory
  - Needs to be autonomous from ground commanding
  - Using simple FIFO schema may result in only downlinking deprecated data – e.g. old SOH data may be irrelevant
  - Possible data management schemas:
    - LIFO – downlink most recent data first
    - Circular queue – oldest data gets overwritten
    - Priority queue – highest priority data downlinked first
    - Expiration – data deleted if not downlinked within allocated timeframe
    - Data type – specific data given downlink priority e.g. (SOH)

UNP Overhead

- Metadata required to transfer and interpret data
- Example telemetry, packet, and radio headers:

Telemetry Header

Mneumonic	Size/bits	Sum of bits
Sync	16	16
Length	16	32
CtlVersion	8	40
Opcode	16	56
TimeDay	16	72
TimeMilliseconds	32	104
RunLevel	8	112
CommandCount	32	144

Downlink packet header/footer

Mneumonic	Size/bits	Sum of bits
Length	16	16
Opcode	8	24
segmentIndex	16	40
segmentCount	16	56
dialogId	16	72
CRC	16	88

Radio Header

Mneumonic	Size/bits	Sum of bits
Protocol Header	16	16
Radio Header	160	176

Margin

- Margin = (1 – percent resource usage) / percent resource usage
- From a software engineering theoretical standpoint margins should be at least 33%
  - As a general rule of thumb, any increase beyond 75% utilization of critical resource requires an exponentially increasing amount of complexity to achieve the desired result (law of diminishing returns)
  - In other words, you have to start investing an ever increasing amount of time optimizing your code in order to get it to run on the limited resource
- Recommendations:
  - Design system with as much margin as possible within budget and size, weight, and power (SWaP) constraints
  - PDR margin goal: ~100% or ~50% resource utilization
  - CDR margin goal: ~33% or ~75% resource utilization

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

UNP Downlink Example

Assumptions:

- Payload – 1 Megapixel camera
  - 1024 X 1024 pixels
  - 16 bits/pixel
- GNC - 200 bytes
  - Collection event GNC - 10 samples/sec
- SOH – 400 bytes
  - Nominal SOH 1 sample/20 sec (3 samples/min)
  - Includes nominal GNC
- Overhead
  - Telemetry header – 18 bytes
  - Downlink packet header/footer – 11 bytes
  - Radio header – 22 bytes

- 1 collection event/day
  - Duration 10 seconds
  - 1 image
- 2 ground contacts/day
  - Link time ~10 minutes
  - Downlink 32 Kbytes/sec
  - Max packet size 1024 bytes

Example Simple Data Budget

- Data Generation over 24hrs

Source	Sample size (Bytes)	Sample Overhead (Bytes)	Frequency (samp/sec)	Duration (sec)	Sample Total	Downlink Overhead (Bytes)*	Total (bytes)
Payload	2,097,152	18	.1	10	2,097,170	67,585	2,164,755
Event GNC	200	18	10	10	21,800	703	22,503
SOH	400	18	.05	86,400	1,805,760	58,194	1,863,954
Total							4,051,212

*** Downlink overhead = (sample total / Max tlm packet size) * (Downlink packet header/footer + Radio header)**

➢ **Link time required = total bytes / downlink rate =**

4,051,212 / ~32,768 = ~124 seconds

➢ **Link time available = (number of contacts / day) * contact time =**

2 * ~600 seconds = ~1200 seconds

Exploration Research and Technology Programs

Trade Spaces for C&DH and FSW

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

The Trade Study

- 4-6 key stages of our trade study approach

1. **General Information & Requirements**
2. **Design & Trade Options**
3. **Quantitative Ranking**
4. **Final Selection**
5. **(optional) Risk Assessments**
6. **(optional) Pitch Deck**

We will follow our internal trade study for developing our multi-spacecraft testbed as a guide

The Trade Study: Phase 1: General Information & Requirements

- Document the **reason or purpose** for the trade study
  - Include expected timeline of the study and resource required (mostly always teammembers)
- Document the expected **results or effect** of the the study
  - Optionally, include the effect of taking no action
- Gather constraints
  - The first resource should be the **Requirements Documentation**
  - Team may include additional constraints
- Phase End Goal is to Develop:
  - **Go/NoGo** Requirements
  - **Objective** Constraint/Requirements
  - **Subjective** Constraints/Requirements

Example Phase 1 Criteria

Go/NoGo
1.[Go/NoGo]: The ADCS shall have a mass no more than 200 grams
2.[Go/NoGo]: [*Binary statement or Requirement*]
3....

Objective Constraints/Requirements
1.[minimize]: Total ADCS mass
2.[maximize]: Total ADCS mounting points
3.[*function*]: [*requirement statement*]
4....

Subjective Constraints/Requirements
1.Mechanical integration: The ADCS has attachment points that are well defined and easily placed.
2....

The Trade Study: Phase 1: General Information & Requirements

Objective Requirements

The following is a list of objective requirements for the PiL selection:

1. [Go/NoGo] The cost of the entire system shall not exceed $200,000.00 USD
   1. Cost per unit less than 200,000 / 100 = $2,000.00 USD
   2. The cost of the entire system is ideally lower than $100,000.00 USD
   3. This metric should include supporting network equipment costs, peripherals, mounting, and ESD equipment
2. [Go/NoGo] The system shuold be capable of running at least 100 networked PiLs
3. [Go/NoGo] The selected PiLs shall be capable of operating cFS
4. [Go/NoGo] The selected PiLs shall be capable of using DDS in their network stacks
5. [Go/NoGo] The selected PiLs shall be capable of interfacing via Gigabit Ethernet for ground testing (on the PCB, via a daughter-board, or some breakout connection)
6. … (there were even more)

Objective Measurements

The following values should be minimized or maximized for later comparison

1. [Min] Total Cost
2. [Min] Total Max Solution Wattage (all boards combined)
3. [Max] Total Number of Space Qualified Boards in Solution
4. [Max] Total Solution Memory (RAM)
5. [Max] Total Solution Storage

Subjective Requirements

The following is a list of subjective requirements for the PiL selection:

1. The selected PiLs shall have a user friendly development environment
2. Good balance of Processor selection(s)
3. The selected PiLs shall be realistically deployable in 2-3 years
4. Expected eases of swarm like deployment
5. Expected ease of virtualization processor selection(s)

Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

The Trade Study: Phase 2: Design & Trade Options

- Primarily a Survey of industry
- List all of your Go/NoGo Statements
  - Start collecting all of your options, receiving quotes, and building a spreadsheet or database
- Enter all possible information
  - Result on failure to receive
- This can be as simple or as complex as necessary
  - We traded single boards, 2-combination boards, and 3-combination boards

1. **N01 - Nvidia Jetson TX2i SoC**
   a. **REF: https://developer.nvidia.com/embedded/downloads#?search=TX2i**
   b. Dev Board:
      i. Astro Carrier Breakout - https://connecttech.com/product/astro-carrier-for-nvidia-jetson-tx2-tx1/
      ii. Optional – NVIDIA TX2 developer kit - https://developer.nvidia.com/embedded/jetson-tx2-developer-kit
2. **N02 - Nvidia Jetson AGX Xavier**
   a. **REF: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/**
   b. Dev Board: Yes – Same device, included
3. **N03 - Raspberry Pi 4**
   a. **REF: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/**
   b. Dev Board: NA
4. **N04 - BeagleBone AI**
   a. **REF: https://beagleboard.org/ai**
   b. Dev Board: NA
5. **N05 - Xilinx Kintex Ultrascale XQRKU060 Space-Grade FPGA**

Non-Combination Matrix (Homogeneous Swarm)

ID	Go / No Go	PiL / Board Option	<= $1,776.18 per unit	cFS compatible	DDS compatible	Gigabit Ethernet	Rad-Hard or Rad-Tolerant / Equivalent	10k TID tolerant	Operating Temp within (–44.0, +81.0)°C ± 15°C
N01	GO	Nvidia Jetson TX2i + Astro Board	YES ($1,054.823)[arrow] + ($183.00)[direct quote] = ($1,237.823)	YES	YES	YES	YES, AFIT TX2i TID report, TX1 Proton Tetsing (just for comparison)	YES, 19-29k AFIT TX2i TID report, but no reboot after 7k	YES, IEEE Aerospace GPU SoC flight computer paper
N02	GO	Nvidia Jetson AGX Xavier	YES ($699.00)[nvidia]	YES	YES	YES	UNKNOWN	UNKNOWN	YES thermal design guide, nvidia forum
N03	GO	Raspberry Pi 4	YES ($76.91)[amazon]	YES	YES	YES	YES similar ARM-v8 processors are close enough	UNKNOWN 160k (model B) TID testing,	YES similar ARM-v8 processors are close enough
N04	GO	BeagleBone AI	YES ($142.80)[Digi-Key]	YES	YES	YES	UNKNOWN	UNKNOWN	UNKNOWN
N05	NO GO	Xilinx Kintex Ultrascale XQRKU060 Space-Grade FPG – Dev Kit	NO ($14,950.00) quote	UNKNOWN	UNKNOWN	NO spec sheet	YES	YES	YES

The Trade Study: Phase 3: Quantitative Ranking

- Time to create a weighted matrices!
- Use measures defined in Phase 1
- Objective is simply Min and Max the defined values
- Subjective has a "scale" in addition to weight
  - 0: Subjectively Poor (bad)
  - 1: Default (normal)
  - 2: Subjectively Great (exceeds)
- Reading tons of spec sheets!

Objective

Category Weights	weight	goal
Total Cost	0.10	Min
Max Total Wattage	1.00	Min
Total Space Qualified Boards	0.10	Max
Total RAM	1.00	Max
Total Solution Memory	0.70	Max

Subjective

Category Weights	weight	goal
User Friendly Dev Env	1.00	2
Good Selection Balance	1.00	2
Future Proof Deployable	1.00	2
Ease of Swarm Deployment	0.50	2
Ease of Virtualization	0.10	2
Ease of IT Integration	1.00	2
Rack Mount Friendly	1.00	2

Design Option (with link)	SCORE	(Normalized) Total Cost	(Nomalized) max total wattage	(Normalized) total space qualified boards	(Normalized) Total RAM in solution	(Normalized) Total Solution Memory
N01	41.833992	0.31	0.888888889	0	0.247678019	0.065530799
N02	60.832525	0.61	0.666666667	0	0.990712074	0.065530799
N03	43.634747	0.96	1	0	0.123839009	0.065530799
N04	39.514009	0.92	1	0	0.030959752	0.0327654
N08	6.1357064	0.44	0	1	0.030959752	0.004095675
N10	40.349554	0.72	0.666666667	0	0.247678019	0.262123198
N19	23.3188	0.73	0.333333333	1	0.123839009	0.065530799
N20	37.413187	0.75	0.888888889	1	0.015479876	0.00819135
C1602-Min	63.865808	0.52	0.677777778	0.05	1	0.167103539
C1602-Max	68.614028	0.01	0.744444444	0.35	0.665634675	0.776539974
C1603-Min	45.606279	0.85	0.994444444	0.05	0.120743034	0.167103539
C1603-Max	61.772844	0.01	0.948888889	0.46	0.095356037	1

Design Option (with link)	User Friendly Dev Env	Good Selection Balance	Future Proof Deployable	Ease of Swarm Deployment	Ease of Virtualization	Ease of IT	Rack Friendly
N01	2.00	0.00	1.00	2.00	2.00	1.00	0.00
N02	2.00	0.00	2.00	2.00	2.00	1.00	0.00
N03	2.00	0.00	0.00	2.00	2.00	1.00	0.00
N04	2.00	0.00	0.00	2.00	2.00	1.00	0.00
N08	0.00	0.00	0.00	0.00	1.00	0.00	0.00
N10	1.00	0.00	2.00	1.00	1.00	0.00	0.00
N19	0.00	0.00	1.00	1.00	0.00	0.00	0.00
N20	2.00	0.00	0.00	2.00	1.00	1.00	0.00
C1602-Min	2.00	2.00	2.00	2.00	2.00	1.00	0.00
C1602-Max	2.00	2.00	2.00	2.00	2.00	1.00	0.00
C1603-Min	2.00	1.00	1.00	2.00	2.00	1.00	0.00
C1603-Max	2.00	1.00	1.00	2.00	2.00	1.00	0.00

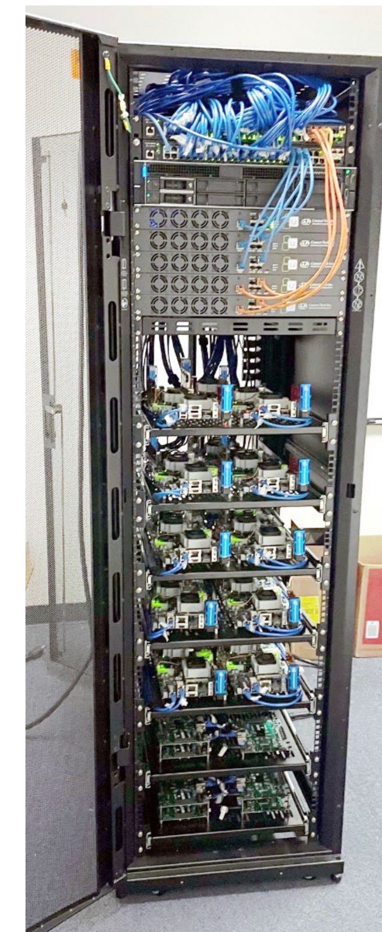
The Trade Study: Phase 4: Final Selection

- Sum the weighted values to reach a score for each design option
  - Then Add Subjective and Objective scores
  - We preferred a 70% Objective 30% Subjective
- At this point, any of the top options are viable within a threshold you are willing to accept (say, +/- 10 points)
  - You have evidence to defend these points
  - If full team was involved from start, all should concur
- Extensively document your final selection
  - Manuals, ICDs, Cost, CAD, User Guides, etc
- Have a backup secondary selection & repeat all options for this
- Develop a “path froward” for using this design

Final Scores

The final scores are calculated with 30% from the subjective score and 70% from the objective score

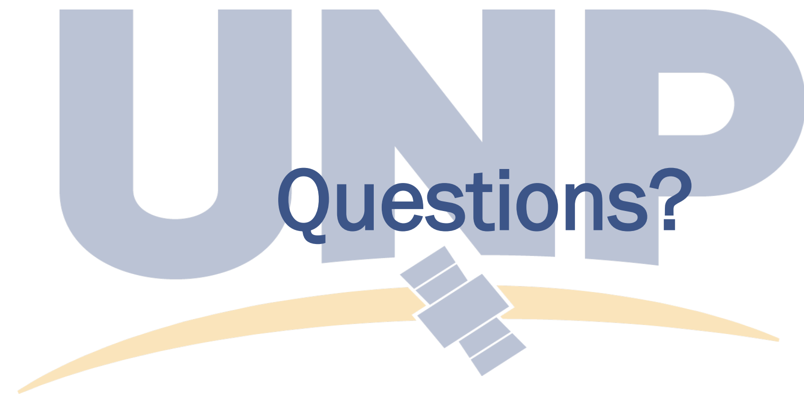
Design Option (with link)	final score
C16020220-Max	70.22
C1602-Max	69.99
C1602-Min	66.67
C16010220	64.11
C160220-Max	63.57
C16010120-Max	61.98
C1604-Max	60.66
C160220-Min	59.92
C1603-Max	59.85
C16022018-51LP	59.54
N02	59.19



Approved for public release; distribution is unlimited. Public Affairs release approval AFRL-2025-2295

The Trade Study: (Optional) Phase 5&6: Risk Assessment & Pitch Deck

- If the project has tracked risks, another matrix can be created
- From the list of best scoring options, you can perform a risk assessment to adjust the scores one last time
- Pitch Deck can also be thought of as “closeout documentation” for a design choice
  - Project or Subsystem Introduction
  - Goals
    - Immediate Goals
    - Long Term Goals
    - Alignment to Funding Agency Goals
  - Proposed Implementation
    - Relevant Alternatives Considered (but not selected)
    - Existing Commercial Solutions (if any)
  - Customers & Stakeholders & Points of Contact on your team
- Q: “Why did you chose to do this?” -> you can have an immediate and satisfying answer for you, your team, and your stakeholders



Questions?