Establishing an Environment for Continuous Integration and Test of Flight Software

Robert Klar, Christopher Mangels, and Randal Harmon

Southwest Research Institute

FSW-09, November 2009

Integration and Test

- Integration and Test is an important part of the development process and represents a significant investment in terms of cost and time
  - In recent organizational estimates, our software integration and testing activities have accounted for more the 40% of the total planned effort
  - Testing tends to be done late in the project lifecycle, limiting its effectiveness
  - Setting up a test environment for a flight subsystem can be challenging
    - Ground Support Equipment (GSE) often is more complex than the flight subsystem itself
  - Latent errors in flight software can be extremely costly

Improving Integration and Test

- Why put emphasis on improving integration and test?
  - Since integration and testing activities represents a large part of the overall planned activities, improving Return On Investment here has a big effect.
  - W. Edwards Deming put forth the notion that “You cannot test quality into a system.”
    - This is true. However, by analyzing and understanding our testing processes, we are able to collect information that can be used to improve our design and implementation processes. This can improve quality.
- Planning for integration and test should be an important part of the software development plan.
  - Espouse the principle of “Design for Test”
  - Hardware/Software resource planning is important (to avoid Marching Army costs)

Continuous Integration and Test

- Continuous Integration
  - Idea that emerged in Extreme Programming community [1]
  - With multiple developers, components for a particular software item are being developed or modified in parallel. Over time, this may result in a very difficult integration ("Integration Hell" [3]).
  - To avoid this problem, integration is done more frequently.
  - A plethora of tools exist today for automating continuous integration.
    - Bitten, BuildBot, CruiseControl, TinderBox, etc.
    - Because there are so many choices, it is important to find one that works well for you.
- Continuous Test
  - Including testing with each integration is a good practice
    - "Test early and often"

Establishing an Environment

- Desirable Characteristics for a CI&T Environment
  - Simple. The system needs to be simple to deploy and use. If not, it will not provide any advantage to improving the I&T process.
  - Reusable. We have many small independent software development efforts. We do not want to “reinvent the wheel” with each one.
  - Cost effective. Since we are involved in many small programs, expensive tools are difficult to procure. Open source tools and low-cost commercial tools are attractive for this reason. Tools with recurring license costs can be problematic.
  - Maintainability. Since space programs need support for many years, we need to make choices to guard against obsolescence.
  - Flexible. Science Instruments and Spacecraft Bus Subsystems have different needs. We need to accommodate both.

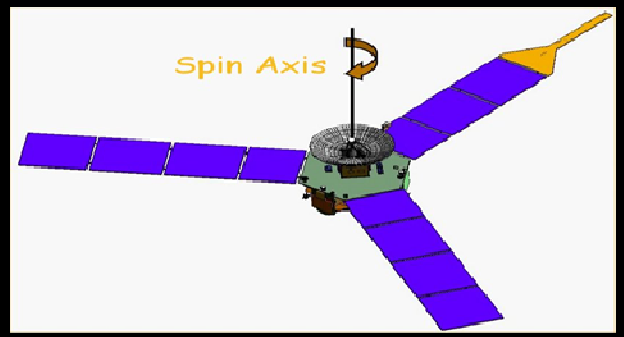
Establishing an Environment

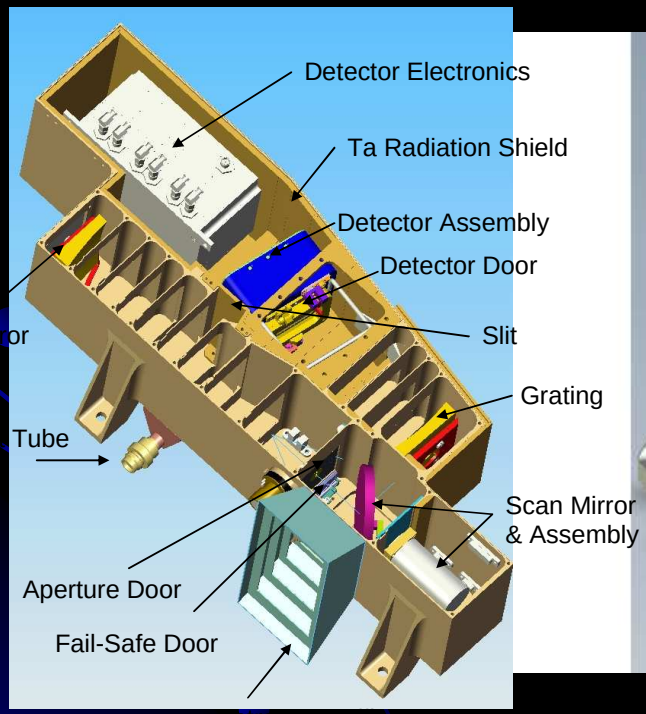
- Plan for testing – do not let it evolve!
  - Resources required for testing are available at different points in the schedule. This is true of most programs.
  - It is desirable to maintain consistency at each level of integration to avoid rework
  - Testing progress needs to be measurable
    - Number of test cases total
    - Number of test cases completed successfully
  - Need to be able to quickly adapt when problems are found
    - Consider backup options for test environments

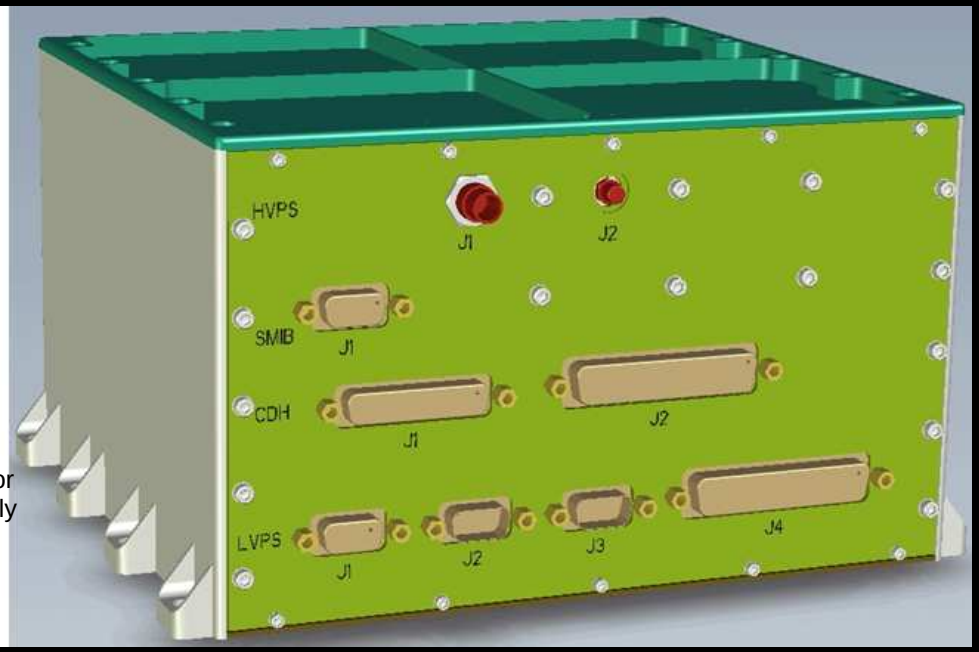
- Setup a regular schedule for integration and test.
  - **Nightly** and **On-demand** were good choices for us.

CI&T Case 1: Instrument (Juno-UVS)

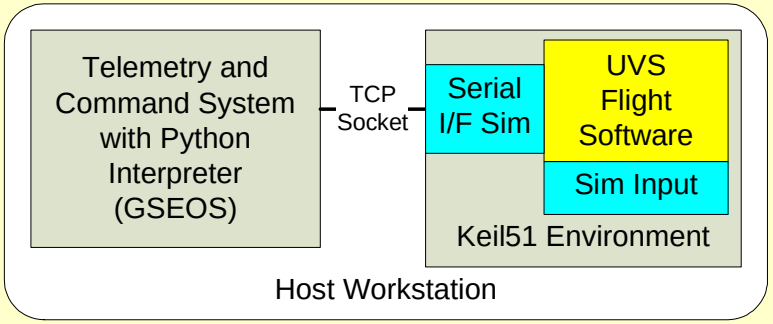
UVS is an Ultraviolet Spectrometer for the Juno Mission (JPL). The spinning Juno spacecraft will perform science observations for about one year around 2017 from thirty highly elliptical 11-day polar orbits around Jupiter. UVS main observations will be performed +/- 3 hours of perijove each orbit.

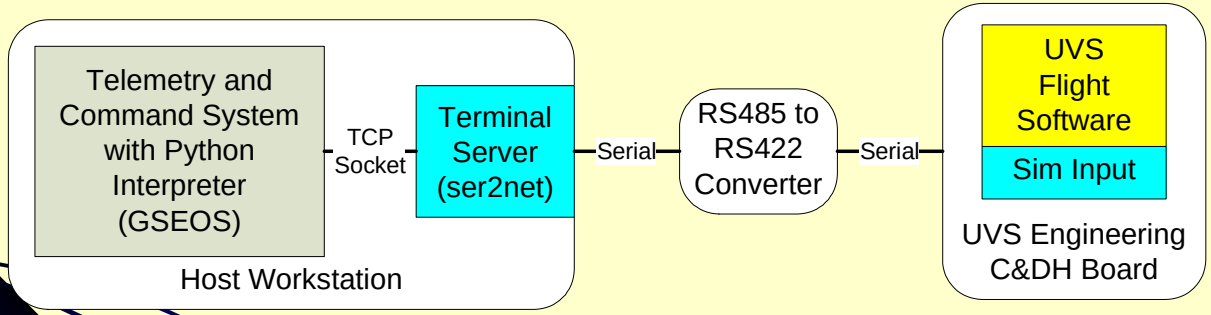


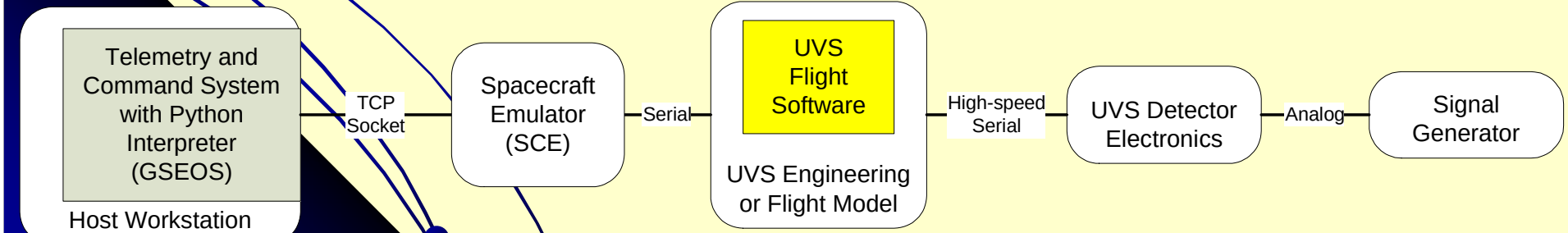




CI&T Case 1: Instrument (Juno-UVS)







- Plan for integration in stages.
  - Using the same Host environment, allows for progression and regression

CI&T Case 1: Instrument (Juno-UVS)

- Build environment is simple
  - C language and make
- Our CI&T environment should also be simple
  - Evaluated CI tools such as CruiseControl and Ant
    - Not a good fit
  - Instead chose to base CI on Python and cron, and CVS
  - Integration and test by “Nightly Script”
    - Rebuilds software and starts execution of Test Scripts
    - Runs until stopped
    - Runs Test Scripts in sequence and out of nominal sequence

```
ITERATION_ORDERS = ['Sorted', 'Random', 'Random', 'Random', 'Sorted']*10

#- Functions ----------------------------------------------------------------
def run_tests(oSeq, order='Sorted'):
    '''run all the tests in the 01XX range'''
    tests = [testmod for testmod in sys.modules.keys() if (testmod.find('tests') != -1) and \
                                                          (testmod.find('.') != -1) and \
                                                          testmod.split('.')[-1].startswith('01')]
    names = [t.split('.')[-1] for t in tests if t.find('.') != -1]
    test_map = dict(zip(names, tests))
    if order == 'Sorted':
        names.sort()
    elif order == 'Random':
        random.shuffle(names)
    else:
```

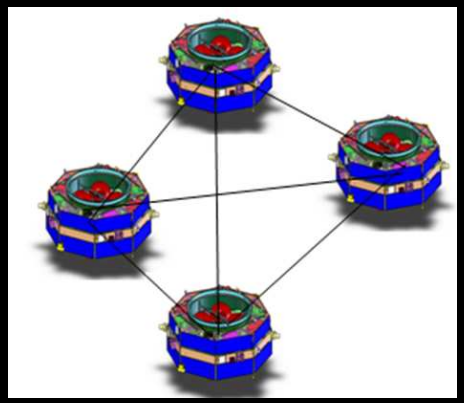
CI&T Case 1: Instrument (Juno-UVS)

- Log files from Nightly Script are analyzed in the morning

```
2009/11/03 03:59:26:TM-HK:   669760: Command accepted: Control Pixel Stimulator
2009/11/03 03:59:26:TM-HK:   669760: Command executed
2009/11/03 03:59:26:TM-HK:   669760: Detector STIM activated
2009/11/03 03:59:26:TSTOK:        : 010: all commands executed
2009/11/03 03:59:26:TC   :        : Control Pixel Stimulator(Stim Off):  0xb3000000
2009/11/03 03:59:34:TM-HK:   669768: Command accepted: Control Pixel Stimulator
2009/11/03 03:59:34:TM-HK:   669768: Command executed
2009/11/03 03:59:34:TM-HK:   669768: Detector STIM deactivated
2009/11/03 03:59:34:TSTOK:        : 020: all commands executed
2009/11/03 03:59:34:TFAIL:   669768: 030: No STIM counts available cannot test event count rates.
2009/11/03 03:59:39:TSEQ :        :
    >>>:==================================================================
2009/11/03 03:59:39:TSEQ :        : >>>: Test Series Summary: nightly_tests has been executing for    92 minutes
2009/11/03 03:59:39:TSEQ :        : >>>: Summary results:
2009/11/03 03:59:39:TSEQ :        : >>>:    56 Failures and    12 Warnings in 33 Tests
2009/11/03 03:59:39:TSEQ :        : >>>: 82.50% tests complete (33 of 40)
2009/11/03 03:59:40:TSEQ :        :
    >>>:==================================================================
2009/11/03 03:59:40:TSEQ :        :
    >>>:==================================================================
2009/11/03 03:59:40:REMRK:        : Attempting to start test sequence: 0160_checkmem
2009/11/03 03:59:40:REMRK:        : Test sequence started: 0160_checkmem
```

CI&T Case 2: Central Instrument Data Processor (MMS-CIDP)

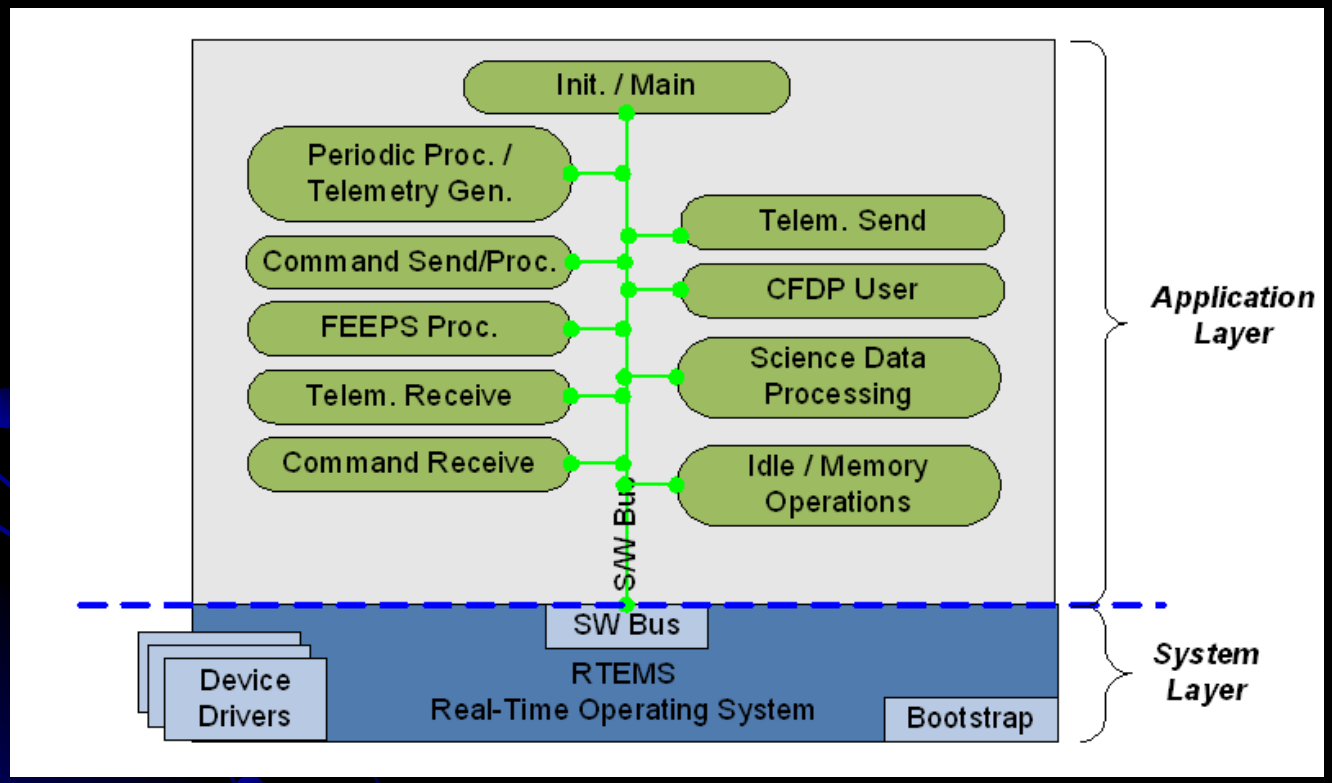
- Magnetospheric MultiScale (MMS) Mission
  - Constellation of 4 identically spacecraft in variably spaced tetrahedron (1 km to several R_E)
    - Ground contacts must be multiplexed in time in order to retrieve data from all 4 spacecraft each day
  - Overall Objectives: To discover the detailed physics of the reconnection process including its controlling factors, its spatial distribution, and its temporal behavior.



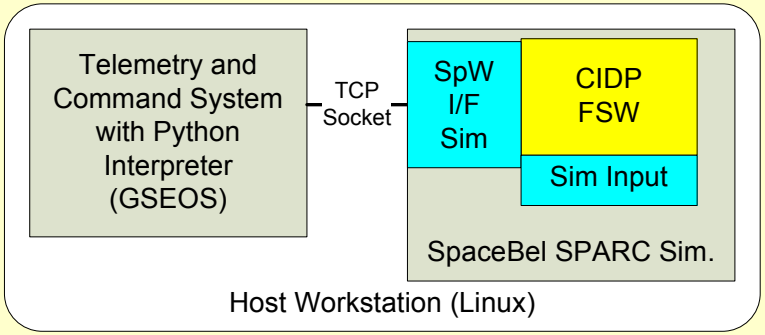
http://stp.gsfc.nasa.gov/missions/mms/mms.htm

CI&T Case 2: MMS-CIDP

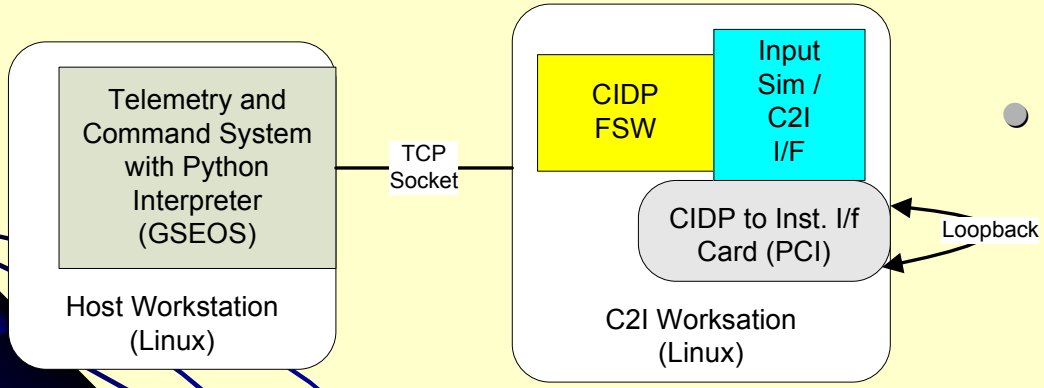
- Layered Software Architecture benefits testing
  - Architecture can be verified in layer
- "Nightly Script" development is in-progress
  - also based on Python



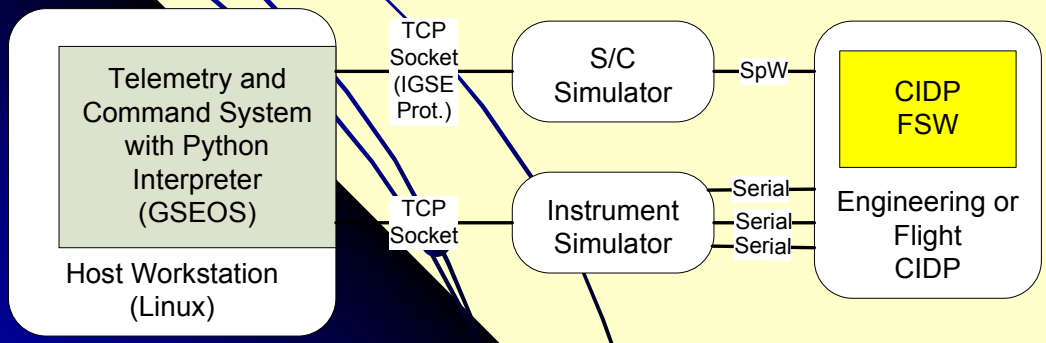
CI&T Case 2: MMS-CIDP



- Plan for integration in stages.
  - Using the same Host environment, allows for progression and regression



- Need to configuration manage test environment software as well as the Flight Software!
  - We use CVS



Challenges Ahead

- Automated Continuous Testing generates large volumes of test data
  - Data analysis tools are necessary to make use of the test results in a timely manner
- Currently test development significantly lags code development
  - “How can I test it if I have not built it yet?”
  - Looking for applications of Test-Driven Development [2]
    - Test Code is developed before a feature is implemented
    - Test is run to verify a FAIL
    - Feature is implemented and test run again
- Currently good quality simulation environments are expensive
  - Adaptations of public domain emulators (Bochs, QEMU, etc.) provide promise to reducing costs here making complete virtual software environments more affordable
- Want to better integrate with Observatory I&T

Questions?

References

1. Martin Fowler. http://www.martinfowler.com/articles/continuousIntegration.html.
2. Kent Beck. *Test-Driven Development: By Example*. Addison-Wesley. 2002.
3. Ward Cunningham. http://c2.com/cgi/wiki?IntegrationHell.