SSC16-IV-1

The Core Flight System (cFS) Community: Providing Low Cost Solutions for Small Spacecraft

David McComas, Jonathan Wilmot, Alan Cudmore
NASA Goddard Space Flight Center
8800 Greenbelt Road Greenbelt, MD 20771
301-286-9038
david.c.mccomas@nasa.gov

ABSTRACT

In February 2015 the NASA Goddard Space Flight Center (GSFC) completed the open source release of the entire Core Flight Software (cFS) suite. After the open source release a multi-NASA center Configuration Control Board (CCB) was established that has managed multiple cFS product releases. The cFS was developed and is being maintained in compliance with the NASA Class B software development process requirements and the open source release includes all Class B artifacts. The cFS is currently running on three operational science spacecraft and is being used on multiple spacecraft and instrument development efforts.

While the cFS itself is a viable flight software (FSW) solution, we have discovered that the cFS community is a continuous source of innovation and growth that provides products and tools that serve the entire FSW lifecycle and future mission needs. This paper summarizes the current state of the cFS community, the key FSW technologies being pursued, the development/verification tools and opportunities for the small satellite community to become engaged. The cFS is a proven high quality and cost-effective solution for small satellites with constrained budgets.

INTRODUCTION

The core Flight System[1] (cFS) is a flight software (FSW) product line developed by the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center's (GSFC) Flight Software Systems Branch (FSSB) over the past 15 years. The cFS product line was developed because previous GSFC FSW reuse efforts had limited success in reducing cost and schedules. Early reuse efforts used a "clone and own" approach where a new project would copy FSW components from one or more previous missions based on functional requirement similarities. This informal source-code based approach to reuse proved difficult for managers to control the scope of the changes and as a result a comprehensive verification and validation effort had to be performed for the new mission which severely limited the cost savings. In addition since FSW components were not configuration managed independent of projects, component quality did not necessarily increase because a single lineage for each component was not maintained.

To meet these challenges the FSSB formed a team of senior engineers to perform a structured heritage analysis across a decade of missions. The initial funding was from non-mission sources which allowed the engineers to participate uninhibited by near-term mission schedules. The diversity of the heritage missions (single string vs. redundant string, varying orbits, different operational communication scenarios, etc.) provided valuable insights into what drove FSW commonality and variability across different missions. The team took the entire FSW life-cycle into consideration, including in-orbit FSW sustaining engineering, as they performed their analysis. Identifying system and application level variation points to address the range and scope of the flight systems domain. The goal was to enable portability across embedded computing platforms and to implement different end-user functional needs without the need to modify the source code. The cFS uses compile-time configuration parameters to implement the variation points. Figure 1 shows the results using a classic software engineering "V-model". The shaded components are cFS artifacts and the <*p*> notation indicates a parameterized artifact. This lifecycle product line approach dramatically increased the number of reusable artifacts and changed how future missions would approach their FSW development efforts.

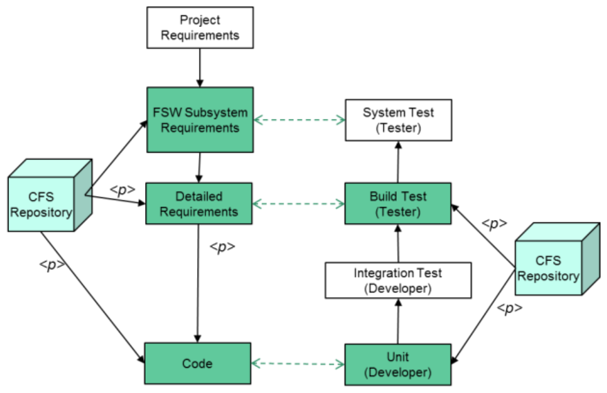


Figure 1: cFS-based Project FSW Lifecycle

ARCHITECTURAL HIGHLIGHTS

While a majority of the heritage analysis focused on FSW functional features a significant and conscious effort was made to address the cFS's architectural quality attributes[2]. Quality attributes are hard to quantitatively trade but they can ultimately determine the success or failure of a software product line. The prominent quality attributes balanced by the cFS include portability, performance, reusability, usability, scalability, interoperability, verifiability, complexity, and predictability. Design meetings, trade studies, and code reviews were used to create a consistent architectural quality attribute balance. Two key trades were performed to determine whether to support file systems and what type of linking to support. At the time of the cFS formulation these were difficult trades because to date no GSFC missions had flown a file system and dynamic linking wasn't supported by RTEMS which was being considered for a mission. The results of the trades were to include file system support and to support both static and dynamic linking. These decision have proven to be vital to the CFS's reusability, usability, and interoperability which has been very beneficial to the ever expanding user base.

Two additional pivotal cFS architectural features are the Application Program Interface (API)-based layers and the definition of an application as a distinct well-defined architectural component. Figure 2 illustrates the four distinct layers and identifies which components have been released as open source. Layer 1 contains the Operating System (OS) and Board Support Package (BSP) and access to the functionality in these components is controlled through two APIs: the Operating System Abstraction Layer (OSAL[3]) and the Platform Support Package (PSP). The OSAL and PSP APIs provide a platform independent (OS and hardware) interface that provides common OS and BSP services. Layer 2 contains the core Flight Executive (cFE) that provides five services that were determined to be common across most FSW projects. The APIs in Layers 1 and 2 have been instrumental in the cFS' success across multiple platforms and the cFE API has remained unchanged since the launch of the Lunar Reconnaissance Orbiter in 2009. Together the APIs define an application runtime environment for the applications[3] in Layer 3. The application layer contains thread-based applications as well as libraries (e.g. linear algebra math library) which can be shared among multiple applications.

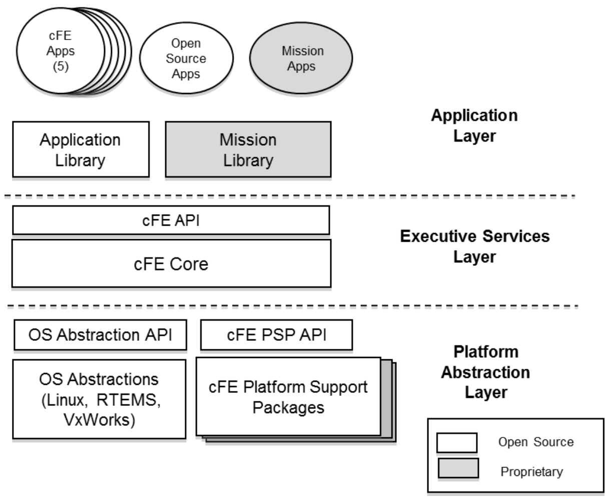


Figure 2: cFS Layered Architecture

The second pivotal architectural feature is the definition of an application as a pluginable component. The cFE enables this feature by providing a core set of services, a runtime environment, and a tool suite for building and hosting flight software applications. The core services include a Software Bus (messaging), Time Management, Event Messages (Alerts), Table Management (runtime parameters), and Executive Services (startup and runtime). The Software Bus provides a publish-and-subscribe CCSDS standards-based inter application messaging system that supports single and multi-processor configurations. Time Management provides time services for applications. The Event Message service allows applications to send timestamped parameterized text messages. Four message classes based on severity are defined and filtering can be applied on a per-message and per-class basis. Tables are binary files containing groups of application defined parameters that can be changed during runtime. The table service provides a ground interface for loading and dumping an application's tables. Executive Services provides the runtime environment that allows applications to be managed as an architectural component.

The cFS manages EEPROM using a file system and uses a script file to determine which application object files should be loaded during initialization. In turn applications subscribe to cFE services during their initialization. Since cFE resources are managed on a per-application basis the cFE supports starting, stopping, and loading individual applications during runtime. This allows applications to be developed independent of the platform, very similar to how apps are managed by smart phones. It can also simplify on-orbit maintenance as demonstrated by the Global Precipitation Measurement (GPM) FSW sustaining engineering team in the fall of 2014 when they successfully replaced the file transfer application without disrupting normal science operations.

CFS COMPONENT SUMMARY

The cFS is a collection of separately configuration managed components. Working up the layers in Figure-1 the configured items are the OSAL, the cFE, and each application. PSPs are developed for specific hardware-OS platforms and are currently bundled with the cFE. Table 1 shows the OSAL platforms currently supported, under development, and being planned. OSAL releases include unit level test suites.

Table 1: OSAL Platforms

Operating System	OSAL Version	Status	Target
POSIX/Linux	4.1.1	Production	Desktop Dev. use CentOS 6.x/Ubuntu 14.04 32 bit
RTEMS	4.1.1	Production	Flying on MMS Mission RTEMS 4.10/Coldfire
VxWorks	4.1.1	Production	Flying on GPM Mission vxWorks 6.4/PowerPC
FreeRTOS	4.2.x	In Dev.	GSFC Dellingr CubeSat Mission FreeRTOS/Arm
VxWorks 6.x SMP	4.3.x	In Dev.	vxWorks 6.7 LEON3 Dual Core
ARINC653	4.3.x	In Dev.	Green Hills Integrity OS
RTEMS 4.12 +SMP	Future	Future	Future Release
Xenomai Linux	Future	Future	Future Release

Table 2 shows the current PSPs delivered with cFE 6.4.2. The level of reuse depends upon a new user's platform similarities. PSP releases include unit level test suites which can be used as starting points when modifying an existing PSP or creating a new one. The cFE is verified at both the unit level and the functional requirements level. All of the unit test source code and functional scripts are part of the cFE release.

Table 2: cFE 6.4.2 Platforms Support Packages

Board/Platform	OSAL Operating System	Status
CentOS/Ubuntu Linux Desktop	POSIX/Linux	Used for development and testing
MMS Custom C&DH Coldfire	RTEMS	1 year in flight on MMS Mission
GPM RAD750	VxWorks	2 yeara in flight on GPM Mission
Gomspace Nanomind ARM CubeSat	FreeRTOS	Under development for GSFC Dellingr CubeSat Mission
GSFC MUSTANG Dual Core LEON3	VxWorks SMP	Under development for GSFC MUSTANG Dual Core LEON3 architecture

Table 3 provides metrics for the cFS as it is being used on GSFC's GPM mission that launched on February 27, 2014. These metrics are representative of the current versions of the cFS components since they have only undergone minor updates since GPM's final build so they provide a good reference point for future missions. Note the EEPROM cFE image and application table images are uncompressed and the applications code images are compressed. Also note memory sizes are dependent upon the configuration parameter settings.

A configuration parameter is defined with either a mission scope or a processor scope. For example the maximum length of an event message is defined at the mission level and whether a local event log is present is defined at the processor level. It's hard to gauge the configuration complexity with simply a number because the parameters span a large functional range from a simple default file name to a system behavioral definition like the time client/server configuration. Note the OSAL and PSP do not have configuration parameters because they are explicitly code for a specific target platform.

Table 3: GPM cFE/Application Metrics

cFE/ App	Logical Lines of Code (non-table)	Config. Parameters	EEPROM (bytes)
cFE	12,930	General: 17 Executive Service: 46 Event Service: 5 Software Bus: 29 Table Service: 10 Time Service: 32	341,561
CFDP	8,559	33	85,812
Checksum	2,873	15	35,242
Data Storage	2,429	27	40,523
File Manager	1,853	22	16,272
Health & Safety	1,531	45	15071
House-keeping	575	8	8.059
Limit Checker	2,074	13	31,026
Memory Dwell	1,035	8	8,617
Memory Manager	1,958	25	15,840
Scheduler	1,164	19	35,809
Stored Command (with 124 command sequences)	2,314	26	104,960

GPM is a NASA Class B earth-nadir pointing mission with articulating solar arrays, a gimbaled high gain antenna, and nearly fully redundant hardware all under FSW control. It has 4 MBs of EEPROM (two duplicate banks of 2MB) and 24 MBs of SRAM. EEPROM is the most constrained memory and the cFS uses ~35% of a 2MB EEPROM bank. From a lines-of-code perspective the cFS accounts for 42% of the GPM FSW (excluding the VxWorks OS). Using the Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM) tuned for NASA missions the cost estimates to develop the complete cFS suite from scratch for a Class B mission like GPM is 49 man years. Using the cFS still incurs costs such as tuning configuration parameters, adjusting task priorities, etc., but these costs have been estimated on the order of 2 man years for a mission of GPM's complexity and class.

CFS COMMUNITY

The cFS was original developed for in-house GSFC missions and is being used on the Lunar Reconnaissance Orbiter launched in 2009, on GPM launched in 2014, and on the Magnetospheric Multi-scale Mission Spacecraft launched in 2015. Over the past few years it has been used across multiple NASA centers including the Ames Research Center's (ARC) Lunar and Dust Environment Explorer spacecraft launched in 2014 and the Johnson Space Center's Morpheus project tested in 2013. The Johns Hopkins University (JHU) Applied Physics Lab's (APL) Radiation Belt Storm Probe launched in 2012 used the cFE and they are also using it for the Solar Probe Plus mission scheduled to launch in 2018. Several NASA missions currently under development are using the cFS and these missions range in scope from JSC's Orion backup computer to GSFC's Dellingr CubeSat. In terms of "Classes" as defined by NASA Procedural Requirements (NPR) 7150.2B[4] these range from Class A to Class D.

In February 2015 GSFC announced the open source release of twelve applications commonly used on most missions which now makes the entire cFS "stack" available as open source. In early 2015 a NASA-wide Configuration Control Board (CCB) with members from six NASA centers (ARC, Glenn Research Center, GSFC, JSC, Langley Research Center, and Marshall Space Flight Center) and the JHU APL was established. The CCB is responsible for reviewing and approving/disapproving the proposed changes to the open source cFS product baselines and technology branches. It also ensures all baseline products meet NPR-7150.2B Class B requirements[4]. This is a critical achievement because each NASA center has a voice in the product's governance which reduces their risks in adopting the cFS and committing resources to products based on the cFS. The CCB currently controls changes to all of the open source artifacts and multiple components of the cFS have been released under the CCB's governance.

Another significant cFS user community expansion opportunity is the recent increase in popularity of CubeSats, cube-shaped nanosatellites that measure about four inches per side and weigh less than three pounds. The NASA CubeSat Launch Initiative[5] provides opportunities for CubeSats to be flown as auxiliary payloads on larger NASA missions. Free rides into space and advancements in sensor, actuator and instrumentation miniaturization allow CubeSats to provide a cost effective solution for technology demonstrations, education research and science missions. As part of the Whitehouse Maker Initiative, NASA is striving to launch 50 small satellites developed by all 50 states within the next five years.

Recent CubeSat efforts have recognized that FSW is one of the big technical challenges because even though the hardware is small the FSW functionality can still be large and complex. Therefore the cFS is positioned as a viable open source FSW solution for CubeSats. As a result the CubeSat community represents a significant potential increase to the cFS user community. However, even though the cFS is open source the interactive cFS community is predominantly within the boundaries of NASA. The remainder of this paper describes recent intra-NASA community activities, efforts to expand the community, and technology initiatives. All of these efforts will benefit the small satellite community.

Within the NASA community the power and benefits of an open collaborative effort have led to several enhancements. Simply expanding the number of projects using the cFS in more diverse usage scenarios has accelerated the product line improvements far beyond what could have been achieved with the limited number of cFS-based projects at the GSFC. A software reuse observation is it takes the following sequence to make software reusable: design component for reuse, reuse the component in a new context (at least 3 is a good sample), and correct the component's initial reuse limitations. Note that all of the cFS applications are on version 2 or greater because they have been through this maturation sequence.

In addition to making incremental changes to the initial cFS artifacts, the NASA community has been expanding the features of the product line. For example, JSC provided a performance analysis tool written in Java that is part of the current cFE open source release. The cFE provides a utility for capturing runtime markers that are saved to a file. The performance analyzer tool creates logic analyzer type graphical displays based on the captured data. This tool is critical for adjusting task priorities and tuning the performance of a new system. JSC also developed generic Command Ingest (CI) and Telemetry Output (TO) applications that are in the NASA open source release process. The current cFS release only contains UDP-based "Lab" versions of the Command Ingest (CI) and Telemetry Output (TO) applications. Users must write their own custom CI and TO applications for flight. JSC's creation of generic CI and TO applications allows the same user interface to be used regardless of the custom transport service layer. This is a substantial step forward and will be of great benefit to the cFS community. New users have found writing their own custom CI and TO applications to be challenging so the generic CI and TO applications will make their deployment of the cFS much simpler and should further expand the cFS user base. These applications also advance the state of the cFS in another significant way. The cFS does not currently have a reusable device plug-in design pattern. The generic CI and TO application designs can serve as models for future applications that need to interface to flight hardware such as sensors and actuators.

Any NASA component released as open source will benefit the global community but we are working on ways to improve the engagement and interaction of the global community. In October 2015 the first cFS Workshop was held at the JHU APL campus that included 11 cFS user presentations[6]. A second workshop is planned for December 2016 at the Beckman Institute at the California Institute of Technology. In June 2016 the University of Florida, who leads the National Science Foundation's (NSF) Center for High Performance Reconfigurable Computing (CHREC) started a cFS website[7]. This website provides discussion forums, news pages, document repositories, and github repositories for collaborative projects. In order to facilitate successful public collaborations and to create a cFS "App Store" ecosystem some technological advancements need to occur.

NASA TECHNOLOGY INITIATIVES

There are several technology initiatives within the cFS community that are focused on streamlining the development process and lowering the barrier to entry in the flight software domain. Like many good ideas, different community members saw the need and were originally developing these independently and were not aware of other similar activities. After the establishment of regular community meetings the community started collaborating. The first technology focuses on the out of box experience for new developers/users. The goal is to create an open source "kit" that contains a development environment with all the elements needed to develop and test the code very rapidly without having to understand all the inner workings. The second technology deals with managing the interfaces, topics, and namespace for the cFS messaging functions. The third technology supports model-based engineering with automatic code generation. The fourth and last technology discussed in this paper automates the testing process such that new platforms and host environments can be used without having to manually rewrite and run regression tests. Each of these is discussed below.

cFS Kits

The cFS was originally developed for GSFC in-house missions and has been incrementally released as open source, therefore it was never packaged as an integrated product line. As a result it can be difficult for new users, especially organizations that have never written FSW to deploy, configure, and extend the cFS for their missions. Two NASA efforts are now underway to create open source cFS "kits" that include a ground system and a spacecraft/environment simulator. These kits provide a complete solution allowing users to immediately have a working product that can be ported to their platform which is much easier than downloading the cFS components and trying to immediately deploy them to their target platform. All users could benefit from cFS kits but the greatest impact and benefits will be to CubeSats and other small FSW teams.

The NASA GSFC Independent Validation & Verification (IV&V) Program is creating the NASA Operational Simulator for Small Satellites (NOS^3). For its operator interface NOS^3 uses Ball Aerospace's COSMOS[8], an open-source user interface for command and control of embedded systems. NOS^3 uses a GSFC open-source simulator called 42[9] to provide spacecraft and environmental simulations. NOS^3 is a sophisticated environment that supports software-only simulations and a hybrid of hardware and software. COSMOS provides enough functionality to be used as the operational ground system so NOS^3 can be used for the entire FSW lifecycle. NASA IV&V is developing NOS^3 as part of its Simulation-to-Flight I (STF-1) CubeSat project in collaboration with the West Virginia Space Grant Consortium (WVSGC) and West Virginia University (WVU). NASA IV&V plans to distribute NOS^3 to other CubeSat developers and release the suite to the open-source community.

The NASA JSC is developing its own cFS kit that is tailored toward providing a cFS training platform. It's implemented within a Virtual Machine, using a custom open-source Eclipse-based user interface and JSC's open-source simulation environment called Trick[10]. The kit is designed for use with a low cost quadcopter drone. The kit contains several lessons and tutorials that each showcase different cFS functionality. These lessons include interfaces with the quadcopter hardware, loading and executing stored command sequences, monitoring telemetry, taking off, landing, etc. NASA JSC is currently in the process of releasing their cFS quadcopter kit as open source.

cFS kits will help expand the cFS community by simplifying the process for new users to learn about the cFS and to deploy the cFS for their missions. The second area of technology initiatives will help all cFS users to participate in a collaborative cFS ecosystem. These initiatives are maturing the capabilities of the cFS' plug 'n play architecture. They are occurring at the application level and the hardware device level. These initiatives are not as mature as the cFS kits but once implemented they will have a significant impact on how user contributions can be integrated back into the product line.

For applications, the goal is to automate the integration of an application to the cFS build, unit verification, deployment, and functional validation procedures. The goal is to have the next cFE release use cmake to control the build process and simplify the manual process of setting and assigning configuration values and messaging topics.

Electronic Data Sheets

The current method of specifying software component and device interfaces is through paper Interface Control Documents (ICDs). This method is manual and prone to human error, and has repeatedly been a source of software errors and system failures. To reduce errors and speed the development process the community is adopting the concept of electronic data sheets originally developed at the Air Force Research Lab as part of the Space plug-and-play (PnP) avionics (SPA) architecture eXtended Transducer Electronic Datasheets[11] (xTEDS). The cFS is using the EDS specification being standardized through the international Consultative Committee for Space Data Systems organization[12].

The EDS concept has a number of use cases that support streamlining the development process as shown in Figure 3. Several of these use cases are in active development at different cFS user organizations with the Component (software) EDS -> Designer tools -> Flight SW components and Component (software) EDS -> Ground System -> Ground System Database flows being readied for integration into the next cFS release.

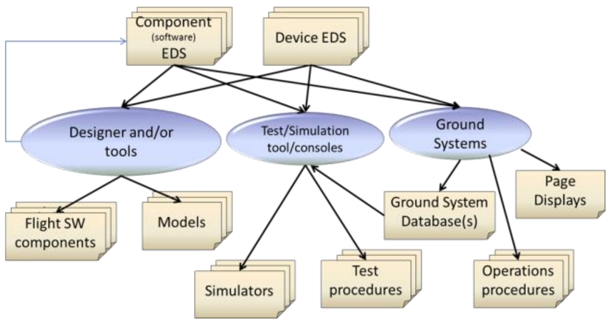
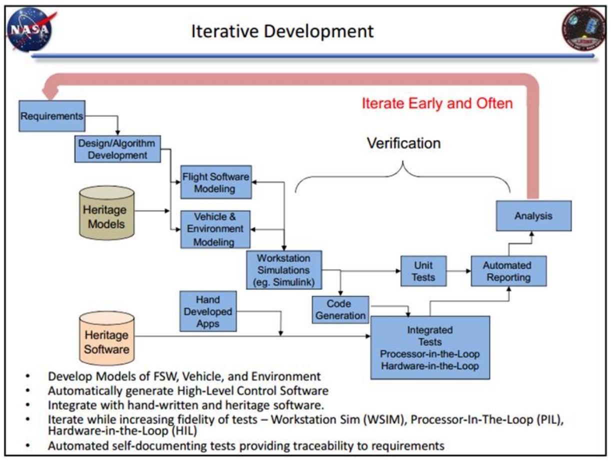


Figure 3: Use of EDS in Software Development

Model-based Code Generation

During the development of the LADEE spacecraft the software team at NASA ARC developed tools to facilitate the automatic code generation of control system Simulink models directly into cFS applications. This tool, called the Simulink Interface Layer[13] (SIL) allowed for rapid/iterative software development supporting an agile approach. The ARC team has provided this tool back to the cFS community and the NASA GSFC is using it on two International Space Station (ISS) instruments: the Neutron start Interior Composition Explorer (NICER) instrument and the Global Ecosystem Dynamics Investigation Lidar (GEDI).



- Develop Models of FSW, Vehicle, and Environment
- Automatically generate High-Level Control Software
- Integrate with hand-written and heritage software.
- Iterate while increasing fidelity of tests – Workstation Sim (WSIM), Processor-In-The-Loop (PIL), Hardware-in-the-Loop (HIL)
- Automated self-documenting tests providing traceability to requirements

Figure 4: Simulink Interface Layer

Assert-based Unit Tests

The UT-Assert unit test framework was created to support automation of unit test execution for software components. Previous unit tests had to be manually reviewed after each run to determine whether the test passed or failed. UT-Assert tests are written with assert statements that evaluate whether a condition is true or false and returns a simple PASS or FAIL that can be passed to a test script. Each test case is written to be self-verifying and eliminates the need for a manual review after each run. For cFS testing a set of stub function libraries have been added to the test framework to support fault insertion. The "white box" fault insertion supports maximum path testing and code coverage. The UT-Assert tests have now been included as part of the automated nightly build process for the NASA cFS Git repository.

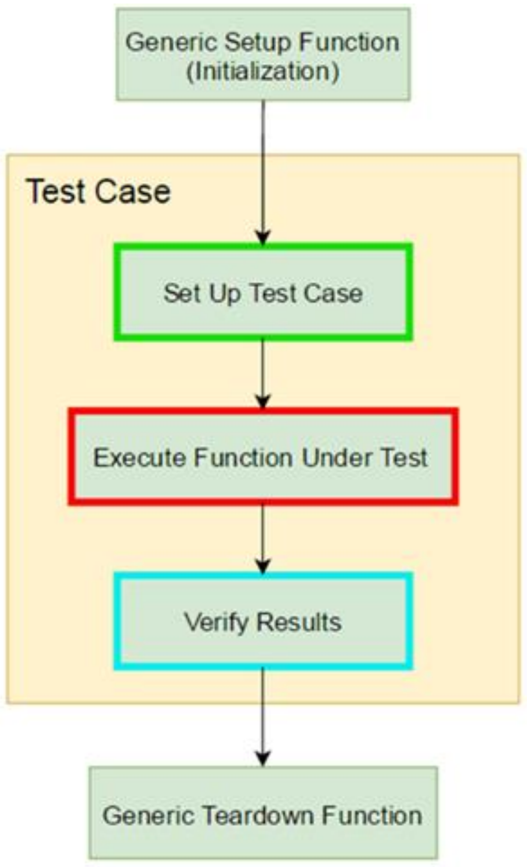

Generic Setup Function (Initialization)
Test Case
Set Up Test Case
Execute Function Under Test
Verify Results
Generic Teardown Function

Figure 5: Unit Test Framework

CONSLUSION

The cFS product line has already shown significant savings in cost and schedule for NASA flight missions while improving the overall quality and usability of the code. With the technology initiatives at or near completion, the cFS and support tools are providing an open source low cost solution for small spacecraft that is being adopted by members of the CubeSat community. As the community widens these initiatives will also continue to build upon themselves and to create the need for new initiatives. For example the cFS kits would be expanded to provide an integrated development environment (IDE) similar to what's provided for developing smart phone apps.

Furthermore as the community starts to supply applications, they may have different levels of maturity. Based on the current community organization we expect applications initially designed by and for the cFS, independent of a project or mission to be rare. We are expecting the majority of the applications to come from either a mission or a technology effort. Mission applications are developed within a mission's budget and schedule constraints therefore they are not typically designed for reuse. Initially they will be submitted "as is" so another mission could still use the application even though it may not be designed for reuse. The reusability maturity process would occur either incrementally or if there's enough demand and funding, an app could be matured as a mission independent effort. The maturity process involves generalizing and parameterizing the requirements, design, and code and updating the unit and build test artifacts to comply with the cFS standards. Applications submitted from a technology effort would follow a similar maturity process except they may not initially be suitable for use as mission critical FSW without some upfront work.

In its current state the cFS is a high quality FSW product line applicable to all classes of FSW. The cFS kits will help expand the cFS user base by simplifying the adoption process. The cFS server hosted by the University of Florida CHREC team, will help cFS users to communicate and coordinate activities. The technological advances towards a more seamless plug 'n play architecture will allow the community to expand and share more applications. Taken together these advancements position the cFS to significantly change how spacecraft FSW is developed and it is especially attractive as a high quality cost-effective solution for small satellites with constrained budgets.

Acknowledgments

The authors would like to acknowledge and thank the cFS community for all the hard work and dedication in maturing the cFS and contributing ideas, applications and tools. We would also like to acknowledge technical and financial support from the NASA Engineering and Safety Center's (NESC).

References

1. National Aeronautics and Space Administration, Flight Software Systems Branch, cFS Overview 2016, http://cfs.gsfc.nasa.gov/Introduction.html.
2. Jonathan Wilmot, Lorraine Fesq, Dan Dvorak "Quality Attributes for Mission Flight Software: A Reference for Architects", http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=454600
3. OSAL, cFE, cFS Applications Suite Open Source SourceForge Download Sites, 2016, **http://sourceforge.net/projects/osal/, http://sourceforge.net/projects/coreflightexec/**, https://sourceforge.net/projects/cfs-XX/ where XX is the application abbreviation. Reference 1 (cFS Overview) has links to all of the applications SourceForge sites.
4. National Aeronautics and Space Administration, Online Directives Information System, Software Engineering Requirements NPR-7150.2B, http://nodis3.gsfc.nasa.gov/displayDir.cfm?t=NPR&c=7150&s=2
5. National Aeronautics and Space Administration CubeSat Launch Initiative, 2015, http://www.nasa.gov/directorates/heo/home/CubeSats_initiative.html#.VWZXks9Viko
6. Core Flight System Workshop: http://flightsoftware.jhuapl.edu/files/_site/workshops/2015/
7. National Science Foundation's Center for High Performance Reconfigurable Computing cFS Website, http://cfs.chrec.org
8. Ball Aerospace COSMOS, http://cosmosrb.com/
9. Eric Stoneking, 42 Simulator, https://sourceforge.net/projects/fortytwospacecraftsimulation/
10. National Aeronautics and Space Administration, JSC Trick Simulation, https://github.com/nasa/Trick
11. Air Force Research Lab Space Plug-and-Play Avionics, http://www.kirtland.af.mil/shared/media/document/AFD-111103-031.pdf
12. CCSDS XML Specifications for Electronic Data Sheets for Onboard Devices and Software Components, 2015, http://cwe.ccsds.org/fm/Lists/Projects/DispForm.aspx?ID=269
13. National Aeronautics and Space Administration , ARC, Simulink Interface Layer, https://ti.arc.nasa.gov/publications/27267/download/