

Cassini Attitude Control Flight Software: From Development to In-flight Operation

Jay Brown¹

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 91109

The Cassini Attitude and Articulation Control Subsystem (AACS) Flight Software (FSW) has achieved its intended design goals by successfully guiding and controlling the Cassini-Huygens planetary mission to Saturn and its moons. This paper describes an overview of AACS FSW details from early design, development, implementation, and test to its fruition of operating and maintaining spacecraft control over an eleven year prime mission. Starting from phases of FSW development, topics expand to FSW development methodology, achievements utilizing in-flight autonomy, and summarize lessons learned during flight operations which can be useful to FSW in current and future spacecraft missions.

Acronyms

| | | |
|--------------|---|---|
| <i>AFC</i> | = | AACS Flight Computer |
| <i>ALF</i> | = | Assisted Load Format |
| <i>ATLO</i> | = | Assembly, Test, and Launch Operations |
| <i>BCIOU</i> | = | Bus Controller, Input/Output Unit (the Bus Controller for the AACS bus) |
| <i>COTS</i> | = | Commercial Off-The-Shelf |
| <i>dof</i> | = | degree of freedom |
| <i>ECR</i> | = | Engineering Change Request |
| <i>FSC</i> | = | Flight Software Change report |
| <i>FSDS</i> | = | Flight Software Development System |
| <i>FSTB</i> | = | Flight Software Test Bed |
| <i>ITL</i> | = | Integrated Test Laboratory |
| <i>J2000</i> | = | Earth mean equator coordinate frame and equinox at year 2000 epoch |
| <i>MIPS</i> | = | Million Instructions Per Second |
| <i>MRO</i> | = | Memory Readout |
| <i>MTA</i> | = | Mono-propellant Tank Assembly |
| <i>OTM</i> | = | Orbit Trim Maneuver |
| <i>PIU</i> | = | Pixel Input Unit |
| <i>RCS</i> | = | Reaction Control System |
| <i>ROM</i> | = | Read Only Memory |
| <i>RTIOU</i> | = | Remote Terminal Input Output Unit |
| <i>RTM</i> | = | Requirements Test Matrix |
| <i>RTI</i> | = | Real-Time Interrupt designated to establish an 8 Hz rate group |
| <i>RTX</i> | = | Real-Time Executive Operating System from TLD |
| <i>RWA</i> | = | Reaction Wheel Assembly |
| <i>SCCS</i> | = | Source Code Control System |
| <i>SID</i> | = | Star Identification |
| <i>SOI</i> | = | Saturn Orbit Insertion |
| <i>SSR</i> | = | Solid State Recorder (mass storage unit) |
| <i>TCM</i> | = | Trajectory Correction Maneuver |
| <i>TRR</i> | = | Test Readiness Review |
| <i>XBA</i> | = | Cross-strap Bus Adapter |

¹ Sr. Software Engineer, AACS FSW Cognizant Engineer, Guidance & Control Software and Flight Software Testing Group, 4800 Oak Grove Drive, M/S 230-104, Member AIAA. E-mail: jay.brown@jpl.nasa.gov

Nomenclature

ΔE = Change in spacecraft energy (km^2/s^2)
 ΔV = Change in spacecraft velocity (m/s)

I. Introduction

CASSINI is the last and largest interplanetary spacecraft built in the twentieth century – arguably one of the most sophisticated robotic missions to date. The end product is a testament to an impressive joint international partnership. Cassini was built and is now managed for NASA by the Jet Propulsion Laboratory (JPL). The European Space Agency (ESA) built the Huygens probe, which piggybacked its way to the second largest moon in the solar system – Titan. The Agenzia Spaziale Italiana built the high gain communication antenna (HGA) which points to Earth practically every single day, since Cassini’s transition to use the HGA¹ on 1 February 2000, to transmit or receive data to or from Earth. Along with the three agencies, a total of seventeen nations contributed to the spacecraft and science instrument designs, fabrications, and analyses.

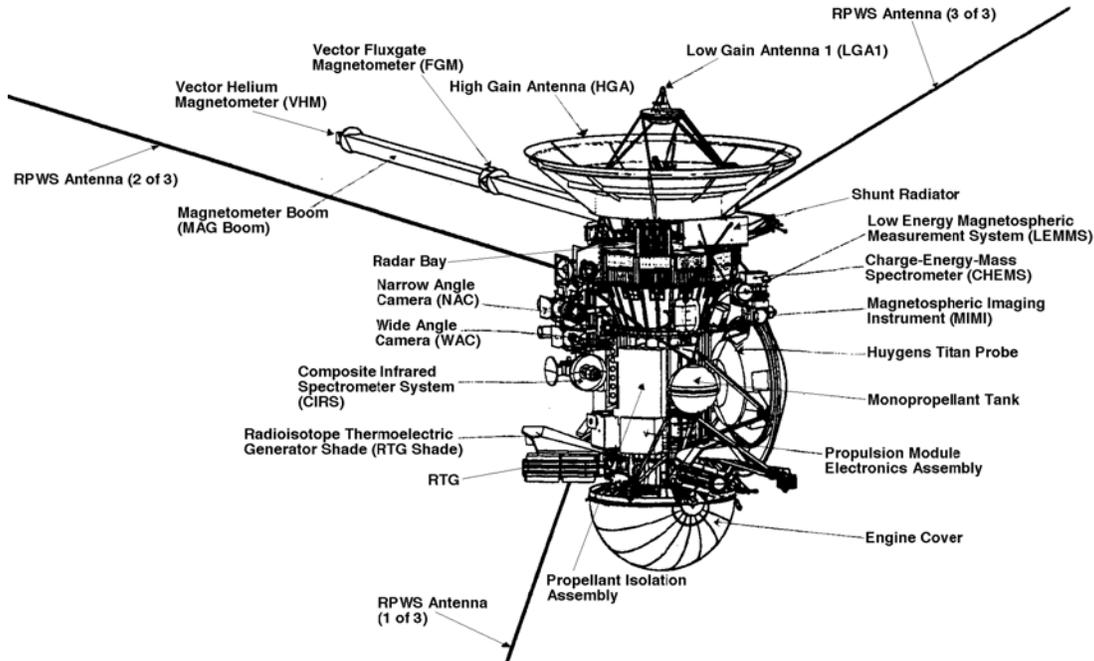


Figure 1. The Cassini Spacecraft. This is the final design of the Cassini-Huygens spacecraft with its twelve science instruments and Huygens probe.² The spacecraft is shown without the hybrid Kapton-Mylar Multi-Layer Insulation (MLI) layer which thermally isolates it in space.

Just as impressive is the testament of symbiotic collaboration between software and hardware. Figure 1 is an illustration of the spacecraft and its instruments. While Cassini stands at 6.8 meters in height and 4 meters in diameter (15 meters with magnetometer deployment) and is constructed with science instruments and engineering attitude control sensors and actuators, it is the software that becomes the glue to control and monitor the entire spacecraft. In particular it is the Attitude and Articulation Control Subsystem (AACS) Flight Software (FSW) which is the scope of this paper.

As an introduction, a brief history of the mission and AACS responsibilities are presented. This paper establishes a timeline of events from initial software consideration to current day software interactions in operations. Key details of the FSW design, development, and test methodologies used to lead up to launch are described. After launch, the operations team took responsibility of Cassini. As planned, AACS FSW continued to be developed and tested to support the interplanetary cruise, critical events at Saturn’s arrival and encounter at Titan, and to meet mission objectives in the Saturnian orbit. A history of in-flight software updates is revealed which leads to lessons learned during FSW development with operations in mind. Maintainability, FSW patching, parameter modifications, and legal range checking are some of the topics of choice. The identification of actions taken throughout the development of Cassini AACS FSW solidifies key software philosophies which aid in the development and maintenance of a successful mission.

II. Mission Background

Cassini-Huygens launched on a Titan IV-B/Centaur in the early morning of 15 October 1997 from Cape Canaveral. This was the beginning of its interplanetary in-flight operations to the sixth planet in the solar system – Saturn. While previous missions were flybys, Cassini is the first to orbit the planet. Cassini is the third heaviest (5,574 kilograms or 6 tons) spacecraft to be launched. Because of the weight, there was not a launch vehicle powerful enough to send it directly to Saturn. To aid in reaching its target and reducing fuel consumption, Cassini flew a Venus-Venus-Earth-Jupiter gravity assist trajectory to boost its velocity to reach Saturn in 6.7 years (this period was considered to be the cruise phase of the mission.) Figure 2 illustrates the interplanetary trajectory design. The prime tour would begin after a successful Saturn Orbit Insertion (SOI) on 30 June 2004. Six months later, Cassini successfully ejected the Huygens probe on 24 December 2004, where it finally reached its destination to Saturn's largest moon Titan on 14 January 2005, and transmitted three hours and forty minutes³ of scientific data to Cassini. The probe data from Cassini was relayed to ESA for analysis. Cassini then embarked on a seventy-four orbit Saturnian prime mission.

Throughout Cassini's four year prime mission, it has had forty-four encounters with Titan as well as three flybys of Enceladus. The surprise discovery of water expulsion from the southern surface region of Enceladus prompted a redesign of the extend mission which was approved in April 2008. Another twenty-seven Titan flybys and seven Enceladus flyby encounters are slated for a two year extension to July 2010.

While it seems like all the excitement started at launch, there were prior decades of ingenious planning and work to get to the launch pad. NASA's Mariner Mark II program began in 1987. In 1989, the first two spacecraft missions were initiated from this program:⁴ Cassini-Huygens to Saturn and Titan, and the Comet Rendezvous/Asteroid Flyby (CRAF). Both missions, with very different payloads and scientific objectives, were to be very similar in design goals. Both spacecraft were to encounter an asteroid with objectives of having many shared components and common designs. These goals initiated an object oriented software approach, to be modular and reconfigurable in design, which are described in the next two sections.

In January 1992, budget constraints were imposed and CRAF was removed from the program. The Cassini design and mission were both simplified. A two-degree-of-freedom articulated High Precision Scan Platform, and one-degree-of-freedom probe relay antenna and continuous rotating turntable were removed from the design. The mission no longer included an asteroid encounter. All instruments and payloads became body fixed, which resulted in a restructure and component simplification of the software (with the removal of articulation architecture.) All instrument, sensor, and antenna pointing would have to now be accomplished by reorienting the entire spacecraft. Simplification has its ramifications in that now, operations sequencing becomes more complex having to balance between maximizing science return and transmission of data back to the ground.

III. AACS FSW Functional Design Background

Several papers have been published describing the AACS FSW design methodology and architecture prior to software implementation and the launch of Cassini.⁵⁻⁶ For continuity, the pertinent highlights will be described and referenced with updated final designs throughout Sections III–VII of this paper. The majority of the proposed CRAF/Cassini articulation (platforms, articulating antenna, and turntable) control aspects of the FSW capabilities no longer existed. The Main Engine gimbals could be characterized in the category of articulation, which were still under FSW control; however with today's standards, this is commonplace and industry would refer to the AACS subsystem as the Attitude and Control Subsystem (ACS). This paper will refer to both acronyms to represent the subsystem.

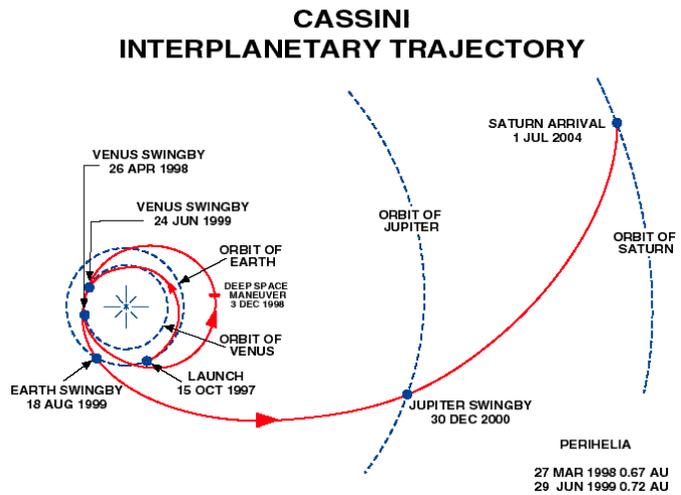


Figure 2. Cassini Spacecraft Mission Timeline. *This time line describes the Cassini trajectory to Saturn from its launch, inner and outer cruise, to the beginning of its prime mission.*

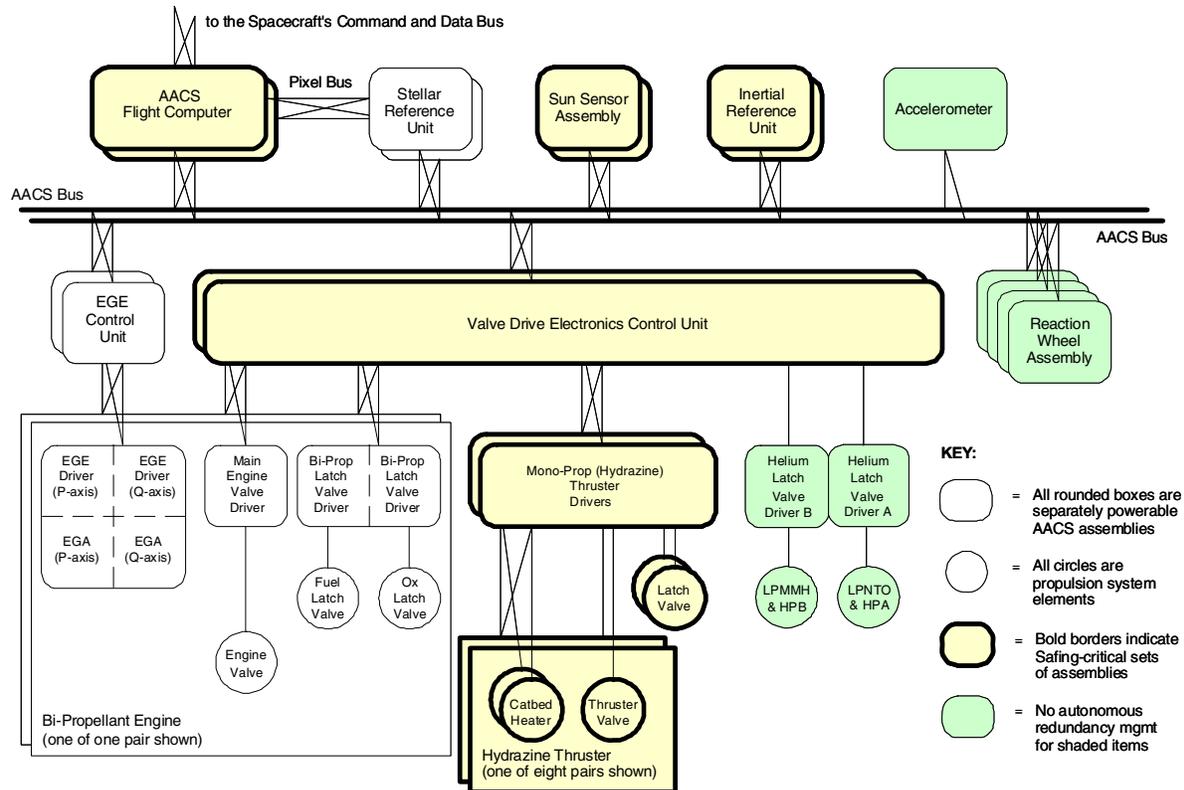


Figure 3. AACCS Functional Block Diagram. Illustrates the ACS-related hardware that needs to be supported by the FSW.

From 1990 to mid 1991, prototype evaluations were performed which resulted in choosing an IBM MIL-STD-1750A 1.28 MIPS microprocessor with 512 kilo-words of Random Access Memory (RAM), 8 kilo-words of Programmable Read Only Memory (PROM), supported by TLD Systems Ada cross-compiler and real time executive (RTX) operating system. These were the beginning steps to support the object oriented design philosophy to address the CRAF/Cassini missions. Early in the design phase, Command and Data Handling Subsystem (CDS) and ACS were targeted to have their own dedicated processors.

CDS is responsible for distributing commands to various subsystems and coordinating all onboard intercommunications. The subsystem collects and packetizes all science and engineering telemetry, manages data to and from two 2.1 gigabit usable RAM Solid State Recorders (SSR), and stores all the onboard command sequences uplinked from the ground.

For ACS, mission requirements mandated a full dual-redundant system which included redundant data buses, prime and backup (block redundant) AACCS Flight computers (AFC), and cross-strapped redundant hardware configurations, as shown in Figure 3. Cassini is a three-axis stabilized spacecraft and maintains control (typically +/- 2 milli-radians deadbanding) using four thruster clusters of four one-Newton RCS thrusters (eight are prime, eight are backup.) For precision pointing control (typically +/- 40 micro-radians deadbanding), three RWAs are used. The prime RWA configuration is three orthogonal (to spin axis) mounted wheels, and the backup RWA is mounted on an articulatable platform. Illustrations of actuator configurations are shown in Figure 4. The Attitude and Control Subsystem responsibilities are to:

- 1) Perform *attitude initialization* by acquiring the Sun, using a two-dof Sun Sensor (SSA), following separation from the launch vehicle, or after a fault recovery.
- 2) Perform *attitude determination* mapping identified stars, using a three-dof Stellar Reference Unit (SRU) and Star Identification (SID) algorithms.
- 3) Perform *attitude propagation* between star updates, using an Inertial Reference Unit (IRU) which contains four Hemispheric Resonator one-dof gyroscopes.
- 4) Perform *attitude control* using RCS thrusters, RWAs, or a 445-Newton gimbaled Main Engine (ME) for propulsive maneuvers.
- 5) Perform control maneuvers to:

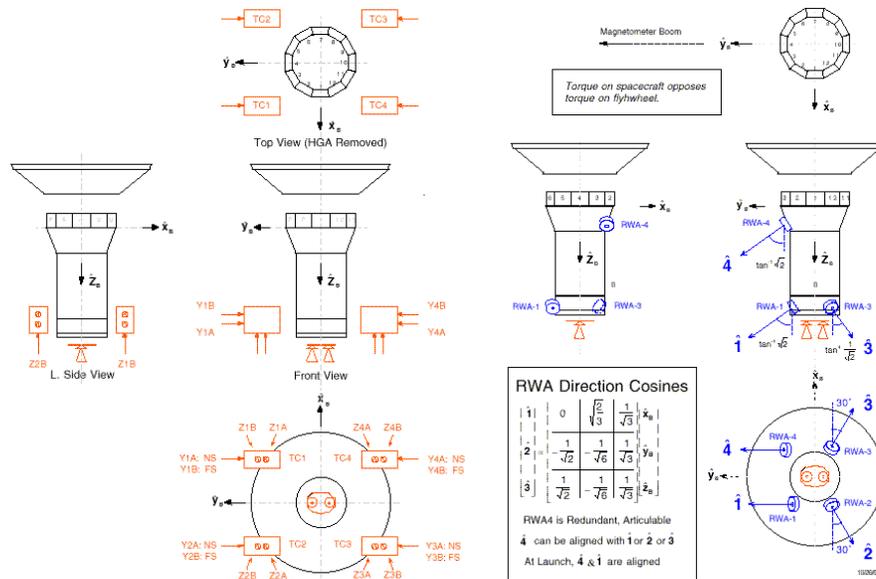


Figure 4. AACS Actuator Configurations. Illustrates the position of RCS thruster clusters, Main Engine gimbals, and RWA positions and alignments.

- a. Point the fixed telecommunications antenna (high gain or low gain) toward Earth; detumble the spacecraft after probe separation; point the high gain antenna toward the probe for probe relay.
 - b. Perform commanded slews (turns) of the spacecraft as required for Trajectory Correction Maneuvers (TCM), science observations, and science instrument calibrations.
 - c. Point instruments on the remote science pallet toward targets that, in general, move relative to inertial space.
 - d. Hold the spacecraft still for probe release and gravity wave measurements.
 - e. Turn the spacecraft at a constant rate about the Z-axis while pointing the Z-axis at the Earth for fields and particles measurement with downlink.
 - f. Point the main propulsion engine in the desired inertial direction during a Main Engine burn.
 - g. Perform axial (Z-axis) TCMs using the RCS thrusters.
- 6) Provide sufficient engineering data in the telemetry stream to support science data analysis and ground support operations.
 - 7) Provide Fault Protection for ACS and certain Propulsion Module Subsystem functions. This involves detecting faults and implementing corrective actions to maintain performance and functionality of the spacecraft.

The Attitude and Control Flight Software is responsible for implementing subsystem requirements in addition to timing, scheduling, and memory functions. Responsible functions are:

- 1) Attitude control.
- 2) Autonomous attitude estimation and inertial vector propagation.
- 3) Star identification.
- 4) ΔV (velocity change) control.
- 5) Message Handling: ACS messages received and transmitted via the CDS bus.
- 6) ACS telemetry.
- 7) Memory usage including loading, mapping, protection and verification.
- 8) AACS hardware configuration and interface management including AACS bus control.
- 9) Timekeeping and scheduling.
- 10) AACS mode control.
- 11) Autonomous fault detection, analysis, and recovery.
- 12) ROM (PROM) functions – Startup ROM which includes many of the memory usage functions, initialization, RAM physical mapping, RAM loading, and load verification.

ACS FSW is a set of two software programs which reside in the AFC: The PROM program and the RAM program. PROM is described above. The RAM program performs all nominal and fault response functionality once loaded and verified in RAM. Program attributes are described in detail in the Software Specification Document⁷ and in this paper.

IV. Software Architecture Background

Rather than continuing to implement the historical software development approaches from past projects, which had software engineers piece together algorithms into flight code and adding fault protection as an afterthought – the architects for Cassini AACS adopted another approach: The concept of interconnected state machines. States capture mode, sequential and dynamic control, state dynamics, and discrete aspects that need to be represented in software. State diagrams provided an avenue of understanding to others outside the software community. This brought together and leveled the playing field between analysts, system engineers, and software developers. Harel statecharts and transition diagrams⁸ were tailored to meet Cassini needs and became a requirement and deliverable to capture detailed design. Since then, variants of Harel statecharts have become a vital part of the Unified Modeling Language. State machines addressed inherent shortcomings of past missions where software functions would use shared or common data pools to communicate or react. The testability of undesirable interactions was of concern with the manipulation of common data. Advancing state in state machines is a well defined process. Allowed transitions are explicitly defined and internal processes related to state history are hidden; which lend well to the object oriented methodology.

Event-driven actions are explicit and lend to achieving synchronization. For Cassini and JPL, all this was a new and evolving concept. A great effort was focused on prototyping software development methodology including personnel interactions, product delivery, and development processes (both software and testbed related). This effort eventually laid the groundwork for the object oriented approach used in the software development process. An object refers to a set of related data and the operations which act on that data. In other words, an object is made up of states where operations (processes) act on its attributes (data) and change its state. The processes of an object are implemented as procedures or subprograms within Ada. The most important paradigm for the object oriented approach is that the data in an object may only be modified by operations of that object where no data are shared that is common. This results in isolation of algorithms and parameters for an object and minimizes coupling between objects.

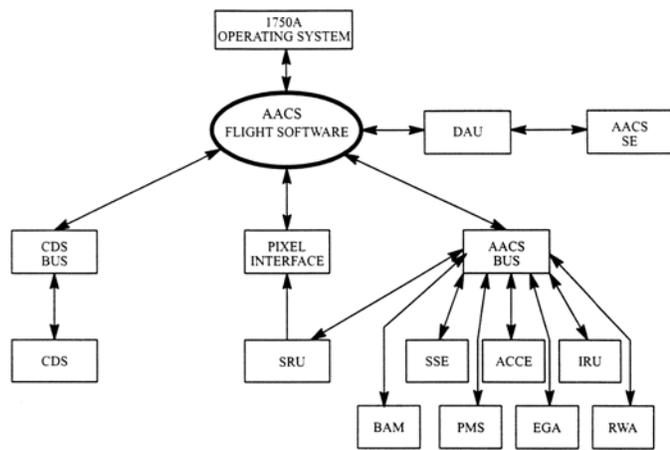


Figure 5. Flight Software Context Diagram.⁵ Software is represented as an oval, lines & arrows indicate data flow, boxes are terminators or external interfaces.

A. Context Diagram Description

Initial architecture design started with the FSW context diagram as depicted in Figure 5. A context diagram simplifies the identification of FSW within the attitude control system. Immediate benefits from these diagrams are that subsystem requirements analysis is aided by indentifying interfaces and dependencies. The FSW has five primary interfaces:⁵

- 1) The 1750A computer and its Operating System.
- 2) The Command and Data Handling Subsystem (CDS) via the CDS bus.
- 3) The Stellar Reference Unit (SRU) via the Pixel Interface and the AACS bus.
- 4) Other AACS hardware via the AACS bus including:
 - a. The Sun Sensors – Sun Sensor Electronics (SSE)
 - b. The Accelerometer – Accelerometer Electronics (ACCE)
 - c. The Gyros – Inertial Reference Unit (IRU)
 - d. The Propulsion Module Subsystem (PMS)
 - e. The Engine Gimbals – Engine Gimbal Assembly (EGA)
 - f. The Reaction Wheels – Reaction Wheel Assembly (RWA)
 - g. The Backdoor ALF Injection Loader – (BAM)
- 5) AACS Support Equipment (SE), for testing, via the Direct Access Unit (DAU) which is disabled in flight. However, DAU memory is used in flight to store the Fault Protection (FP) event log and post-mortem data.

The primary interfaces also identified the next level of interfaces which were necessary since both buses and hardware support had to be represented in FSW. The AFCs are remote terminals on the MIL-STD-1553B CDS bus, which can support up to 31 remote terminals. Each AFC receives ground commands via the CDS bus. The prime AFC interfaces with its peripheral hardware on a dedicated MIL-STD-1553B (electrical standards) AACS bus with a custom protocol that can support up to 255 remote addresses and bus transactions of up to 255 data words.⁹

There are three bus interfaces:⁵ The CDS bus interface, the Pixel interface, and the AACS bus interface. The CDS bus manager sets up and maintains the protocol to the Bus Interface Unit and handles handshake interrupts. The Pixel Interface Manager sets up the Pixel Interface Unit (PIU) to store pixel data to an assigned address, enables writing to PIU memory, and services transmission error interrupts. The AACS bus manager accepts packets from the hardware managers and prepares transmission packets for the bus. It handles packet transmission and receipt from memory shared with the Bus Controller as well as generating and servicing handshake interrupts. Reply packets from the hardware are distributed to the individual hardware managers.

B. Level 0 Architecture Diagram

The highest level of architecture diagram is the dependency diagram, as illustrated in Figure 6. Early in the design process, control calls are not known and the Level 0 diagram helps establish hierarchy. At the top is the external operating system which links to the FSW executive (FSX) as a result of an interrupt, system startup, or via system calls. FSX services interrupts, handles timing services, schedules all execution, initiates each object's execution, and performs tasking. Telemetry, XBA Manager, Fault Recovery, Fault Analyzer, Command Handler, Configuration Manager, and Mode Commander are specific high-level FSW objects identified to fulfill functions required by lower-level objects. The objects share no data. Only global type definitions and constants resolved at compile time are shared.

To highlight a few FSW objects, the XBA is the AACS interface to the CDS bus which supports both receive and transmit transactions. Telemetry can fetch telemetered data from any object which may be sent to CDS by interfacing with the XBA Manager. Faults may be raised by any object and sent to the Fault Analyzer. If recovery involves commands, fault commands take precedence over CDS commands. Commands from CDS, routed through the XBA Manager, or Fault Recovery commands interface with the Command Handler for processing. The Command Handler determines command priority, conflict resolution, and routes the proper commands to the Mode Commander (Software Configuration Manager) or Hardware Configuration Manager for execution.

C. Level 1 Architecture Diagram

Level 1 architecture is also referred to as the control architecture.⁶ The Mode Commander maintains the software configuration and allowable changes in the software configuration (mode changes). A detailed description of mode transitions is captured in Section VII. The Mode Commander sets goals for attitude control and manages the states of attitude determination. Figure 7 illustrates a three layer hierarchy: The top level commander, mid level controller, and low level hardware managers. ACS functionally has two primary goals: Attitude determination and Attitude control.

Attitude determination is comprised of the Attitude Estimator and Star Identifier (SID), and estimates current attitude on data received by ACS hardware sensors via hardware managers. Attitude error is computed with the cycling of data between determination and control and comparing current attitude with commanded/desired attitude.

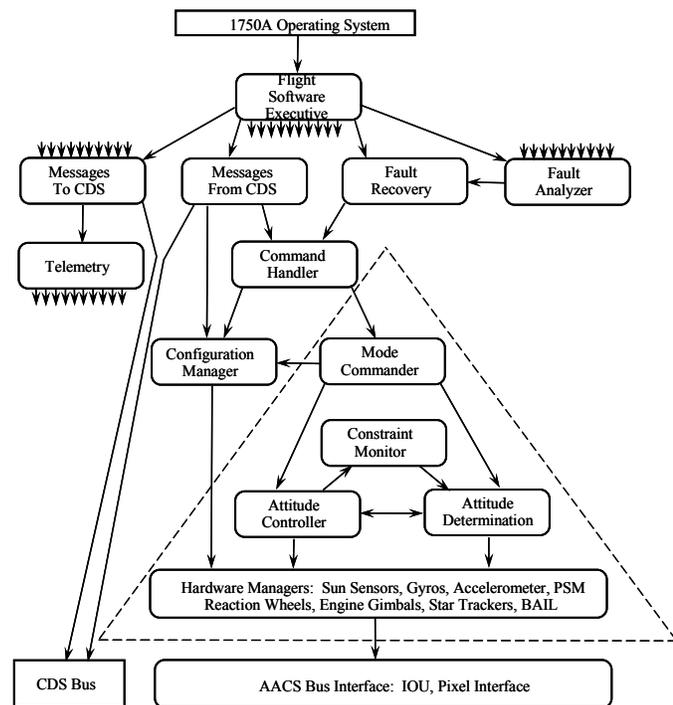


Figure 6. Level 0 Flight Software Architecture Diagram.⁵ Highest level FSW representation. Rounded rectangles represent objects, rectangles represent external interfaces or terminators, arrows indicate direction of dependency, arrowheads indicate source or sink dependencies on all objects.

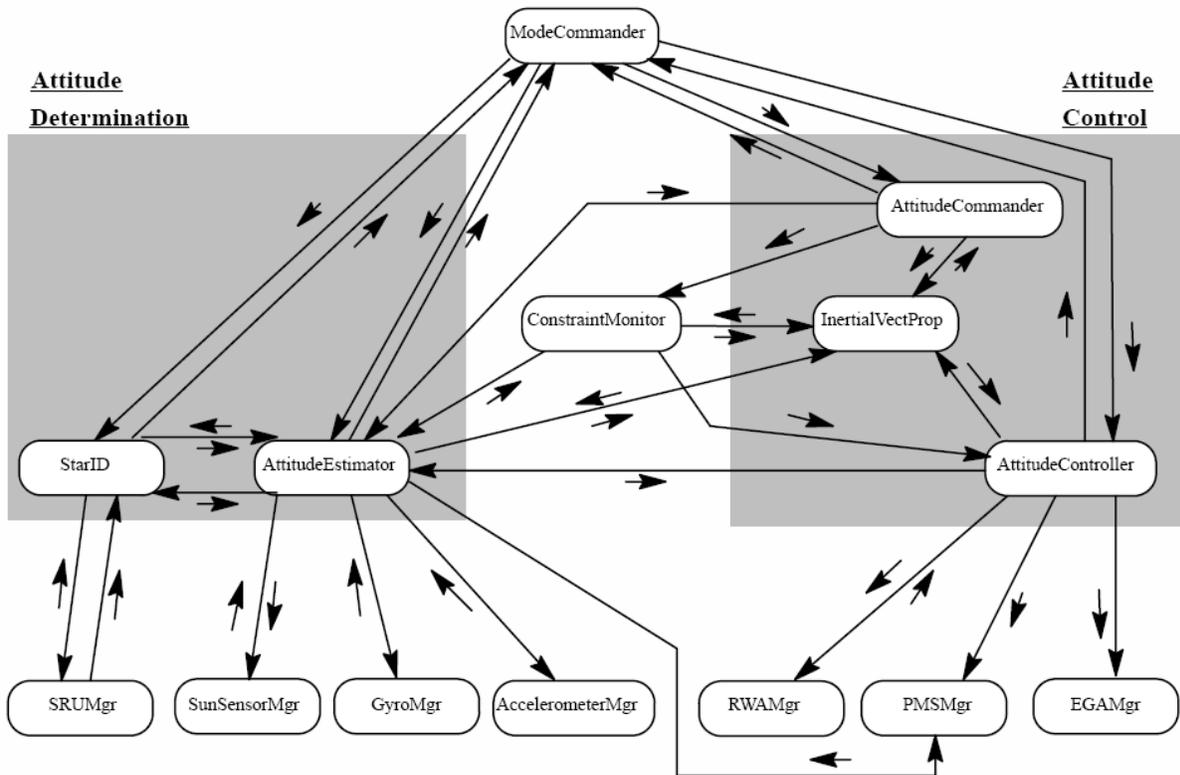


Figure 7. Level 1 Flight Software Architecture Diagram.⁶ The next level of architecture to show control by indicating direction of event call functions between objects. The small arrows indicate direction of data flow.

Attitude control is comprised of the Attitude Commander, Inertial Vector Propagator, and Attitude Controller. The Attitude Commander generates attitude command profiles (rate, position, acceleration) for pointing. The Inertial Vector Propagator models time-varying inertial vectors in the inertial frame, and also serves as the repository of spacecraft fixed vectors, i.e. boresights, of interest. The inertial vector and body vector tables are configurable. The Attitude Controller responds to commands and controls the spacecraft attitude. Control is performed using ACS equipped actuators made up of RCS thrusters, reaction wheels, or Main Engine gimbals.

There is a governing Constraint Monitor which checks estimated and desired attitudes for geometric violations that may potentially harm the spacecraft payload such as prolonged Sun exposure.

D. Object and State Transition Diagrams

From the Level 1 architecture diagrams, the lowest level nodes are termed an object. Object diagrams depict detailed control and data flows not captured in the Level 1 diagram.

E. State Transition Diagrams

State transition diagrams capture both state and data flow in one diagram. A state transition occurs when an event labeling that transition occurs. The procedure and function calls represented in the object diagram are the events of the state transition diagram. The event triggers the state transition and continues to completion without interruption. Datum is represented by a state variable allowing the variable to take on an arbitrary number of states. A state can host an entire state machine within it by partitioning. Nested state machines can run in parallel with machines in other partitions.

V. ACS FSW Development Strategy

Cassini FSW development and testing for the ACS kept to strategies described in JPL's legacy D-4000 Software Management Standards Package. Cassini FSW requirements and practices have influenced JPL's current Software Development Requirements (SDR) plan and Flight Project Practices. These documents describe the "whats" that are needed for software development. Currently, JPL software development is going to Software Development

| NASA Phases | COMMITMENT | | APPROVAL | | LAUNCH | | | |
|-----------------------|------------------------------|---|-----------------------------|--------------------------|--|---|-----------------------------------|---------------------------------------|
| | FORMULATION | | CONTRACT | | IMPLEMENTATION | | | |
| JPL Life Cycle Phases | Pre-Phase A: Concept Studies | Phase A: Concept Development | Phase B: Preliminary Design | Phase C: Detailed Design | Phase D: Fabrication, Assembly, Test & Launch Operations | | Phase E: Operations & Sustainment | |
| Major JPL Reviews | Mission Concept Review MCR | Preliminary Mission & Systems Review PMSR | Project PDR | Project CDR | Assembly, Test & Launch Readiness Review ARR | Pre-Ship Review, Operation & Mission Readiness Reviews PSR, ORR & MRR | Post Launch Assmnt Review PLAR | Critical Events Readiness Review CERR |

Figure 8. JPL Life Cycle Phases. Phase development also applies to Cassini, however considering Cassini is a legacy project, some review names and requirements have evolved to meet today's standards.

Standard Processes (SDSP) which allows tailoring of project development goals to comply with the SDR. The SDSPs describe the “hows” to develop software. Figure 8 identifies JPL’s project life cycle phase development. Cassini is considered a legacy JPL project which met most if not all of JPL’s current development standards. Some documentation and review requirements were different, but Cassini’s philosophy of design, development, and testing were consistent with today’s standards and frameworks.

For many missions, FSW development ends at launch, where it then goes into a maintenance or sustainment phase. Cassini ACS FSW design and development took on a two phased approach. For the first phase, in the allotted time, FSW code would be ready for launch and early cruise operations. The second phase involved outer cruise, critical sequence, and prime mission support FSW development, which would be deferred and uplinked in flight during the 6.7 year cruise phase of the mission and beyond.

VI. Pre-launch Phase FSW Development, Implementation and Test Methodology

Cassini ACS FSW was based on an incremental development model as shown in Figure 9. The object oriented development approach created an environment with minimal and clearly defined interfaces which improved future design and integration efforts. FSW development and testing spanned over six years prior to launch. Modifications to ACS FSW are ongoing to meet current mission objectives for the prime and extended mission phases, which are explained in Section VIII. These methodologies will reveal the FSW development and test philosophy used to establish one of the most successful interplanetary missions to date.

A. Pre-launch FSW Development Methodology

Software development officially starts after the Preliminary Design Review (PDR), which marks the beginning of the Implementation phase of a project’s lifecycle. Within the Implementation cycle are Detailed Design; Fabrication, Assembly, Test & Launch Operations; and the Operations & Sustainment phases. These phases are commonly known as Phase C, D, and E, respectively at JPL. Cassini is now in Phase F which is designated for closeout. Cassini refers to this phase as extended mission operations and support.

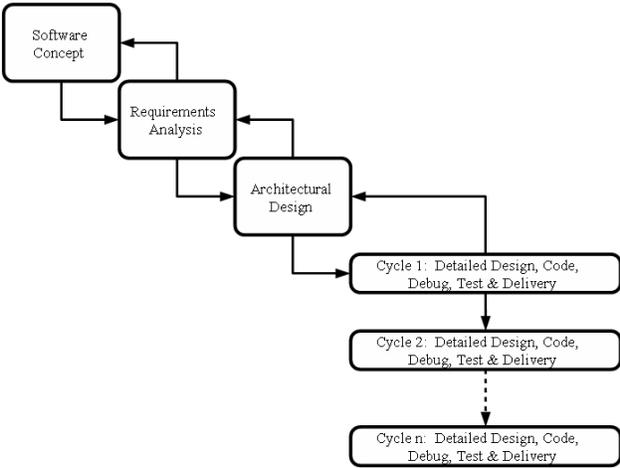


Figure 9. Incremental Development Model. Also known as a multi-build model, the first build has a subset of planned capabilities and following builds add capabilities. Cycles can repeat without deliveries to operations. Flow-up arrows represent feedback from prototyping.

For CRAF/Cassini, the PDR was held in November 1991. Due to project descope activities, a delta Cassini (without CRAF) PDR was held in November 1992. The Critical Design Review (CDR) was held in December 1993. Incremental software builds were released from September 1994 to October 1997. Each build entailed detailed design peer reviews, software design reviews, code peer reviews, unit test plans and testing, unit test peer reviews, integration and acceptance testing, and test readiness reviews before delivery to system-level integration and testing. Figure 10 can be referenced that illustrates the implementation flowchart. Software development guidelines were established which identify, for each review in the build strategy, the focus of the review, materials needed to be reviewed, needed participants, and actions taken following action item closure. All processes were adhered to and repeated when necessary.

Since FSW development and deliveries were based on the incremental development model, the incremental FSW builds were separated and identified as follows:

- 1) Builds A1 established the skeleton structure.
- 2) Builds A2 introduced Control Algorithm capabilities.

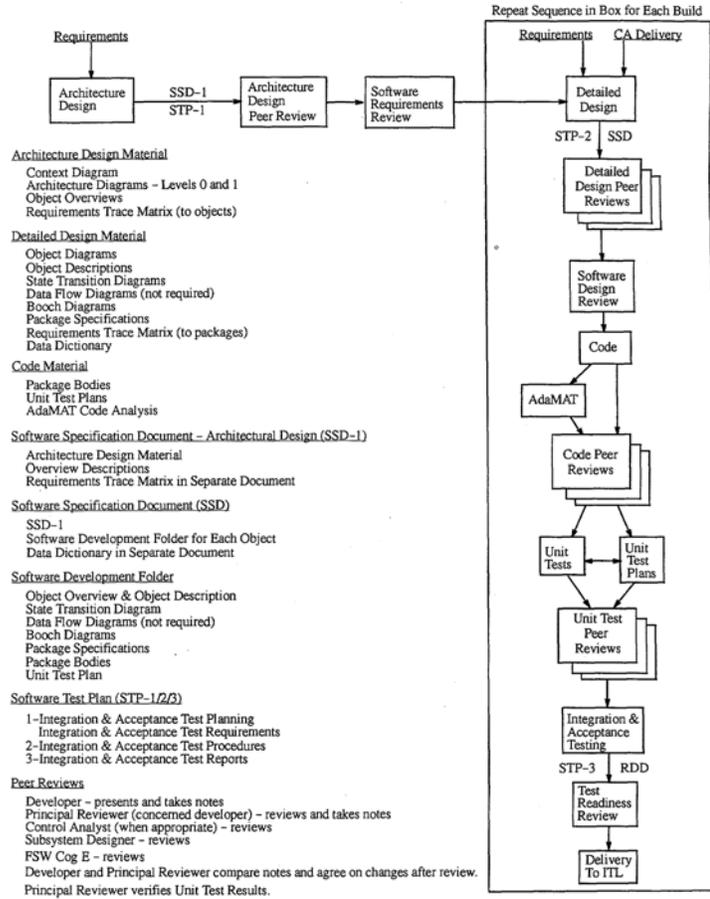


Figure 10. FSW build strategy for development and test. ⁵ This is the final build strategy established by early prototyping.

Table 1. ACS FSW Pre-launch Builds

| FSW Version | Capability | Key Update Description | FSW Testbed* | Freeze | ATLO* |
|-------------|--|--|----------------------------------|--------------------|--|
| A1 | Skeleton | <ul style="list-style-type: none"> These builds contain the FSW executive (operating system interface) which includes the rate group architecture, limited command processing and telemetry, hardware and software interfaces, device drivers Control algorithms and other mid-level processing were stubbed out | A1 (10/2/94) | - | - |
| A2 | Control Algorithms | <ul style="list-style-type: none"> These builds contain all control algorithms and any upgrades to build A1 A2f added RAM Verification Block, SID attitude initialization, IRU and ACM functionality, SID track descope A2f+ added SID attitude initialization, telemetry rates, and mini-functional Fault analysis and recovery were stubbed out First builds designated to be delivered to ATLO | A2f (10/18/95) A2f+ (1/15/96) | 11/27/95 4/1/96 | 1.0 (1/15/96) 1.1 (4/22/96) |
| A3 | Fault Protection | <ul style="list-style-type: none"> These builds contain preliminary Fault Protection (FP) and any fixes to build A2f+ Added fault monitors, FP Analyzer, and FP Response | A3 (2/19/96) | - | - |
| A4 | Fault Protection and Constraint Monitoring | <ul style="list-style-type: none"> These builds contain final Fault Protection (FP) designs and any fixes to build A3 A4f added Constraint Monitor, SID track, more functionality to FP Analyzer and FP Response. Supports TCMS A4f+ contained FP upgrades. Supports Operation Modes (pointing with RWAs) | A4f (4/8/96) A4f+ (6/1/96) | 7/8/96 8/5/96 | 2.0 tcm (7/22/96) 2.0 opm (8/19/96) |
| A5 | Full Functionality | <ul style="list-style-type: none"> A5 Added Backdoor ALF Injection Loader (BAIL) functionality and 7Parameter command capabilities and fixes to build A4f+ A5+ contained bug fixes to build A5 | A5 (12/16/96) A5+ (3/31/97) | 2/10/97 4/14/97 | 3.0 (2/20/97) 3.1 (4/28/97) |
| A6 | Launch Load | <ul style="list-style-type: none"> Parameter updates only to build A5 A6.4.15 (launch load) on default SSR partition, A6.5.6 (cruise load) on non-default SSR partition | A6 (6/2/97) | 6/16/97 | 4.0 (6/30/97) |

* - FSW Version (Release Date)

- 3) Builds A3/A4 integrated Fault Protection (FP).
- 4) Builds A5/A6 were full functionality builds. The BAIL functionality was added in A5 to address deep under-voltage AFC recovery, and contained other fixes to FSW. Build A6 was the ACS FSW launch load.

Table 1 captures key FSW build capabilities.

B. Pre-launch FSW Test Methodology

It is common that software testing begins before PDR by developing the test requirements and plan. After PDR, FSW tool development begins and test cases are defined which also includes initial unit testing. After CDR, unit and interface testing begins and continues to test FSW functionality and interface verification and validation. This testing approach is used to support Test Readiness Reviews (TRR) and deliveries to Assembly, Test, and Launch Operations (ATLO). Test milestones¹⁰ are: Requirements Trace Matrices (RTM) development, test plan development, test case design, test case and procedure development, Test Readiness Review, test case execution, test anomaly reporting, test analysis, and test result reporting.

Essential to the test program was the Software Test Plan. The mode and state driven architecture established the method for how unit testing was performed and what the verification methods would be for pass/fail criteria. Unit testing involved low-level verification of functions, monitoring of state or mode transitions, and boundary condition testing. Future FSW interfaces were stubbed (simulation of called components not yet implemented) to allow the proper verification of each unit test. Interfaces were un-stubbed once they became available. All unit test results were peer reviewed for content, functionality validity, and results.

Concentration was on testing interfaces between objects and requirement conditions specified in the AACS Functional Requirements Document, Software Specification Document, Interface Control Document, RTMs, safety requirements, and fault protection requirements. Requirements could then be traced to the specific scenario or function tests. Integration testing was tied to the planned number of FSW builds. The integration testing process was divided into phases and was repeated for each build. Integration tests comprised of four phases:¹¹

- 1) Phase 1 (Package Specification Testing): This was a skeleton of specification stubs constructed and compiled together. This was the first test of interface definitions.
- 2) Phase 2 (Package Body Testing): Using the FSW executive, the code specifications and bodies were compiled together. This was the second test of interface definitions.
- 3) Phase 3 (Informal Thread Testing): This phase was accomplished in parallel with phase 2. As more objects were added, more functionality could be tested. This controlled increase in complexity as well as the testing of new interfaces and functionality was achieved using an environment which allowed testing of individual threads.
- 4) Phase 4 (Scenario and Functional Testing): Using a full set of integrated objects, scenarios and functional tests would test the full functionality of the build to validate the implementation of prescribed interfaces and required functionality. The distinction between scenario and functional tests was that the scenario tests were based on inputs from the AACS systems engineering group, and the functional tests were developed by the FSW group without such inputs. Both sets of tests were developed in the form of test scripts.

At the completion of each integration test phase, an integration test baseline was established. The baseline was configuration controlled which included tested software, test drivers, and test stubs. Regression testing consisted of rerunning selected unit test for the objects and rerunning selected scenario and/or functional tests from phase 4 of the integration testing.

Acceptance testing started with the verification of the RTM to the requirements for FSW. Any requirements not validated by scenario testing had one or more functional tests developed to validate them. The final approval of acceptance testing was the successful completion of phase 4 scenario and functional testing.

Just as important as testing were the reviews for test readiness and reviews of test results. TRRs marked the release and acceptance of FSW for integration testing. Problem reporting databases were established with the AACS subsystem-level Flight Software Change (FSC), system-level Problem Report, and system-level Engineering Change Request (ECR) database documentation systems. The ATLO Readiness Review (ARR) marked the release and acceptance of the final FSW build for system-level testing. AACS subsystem test plans were integrated into system test plans. RTMs, test trace matrices, test plans, and test cases were all utilized to test and validate interfaces in the spacecraft system environment. ARR testing and reports led to the support of Operation Readiness Tests and the final review before launch, the Operations Readiness Reviews. This review established the readiness of ground and flight support, facilities, plans, processes, procedures, and verification & validation status to meet the mission objectives.

VII. Keys to Final Design and Implementation

While there are hundreds of examples that make the ACS FSW noteworthy, there are key outstanding attributes that have really contributed to its success:

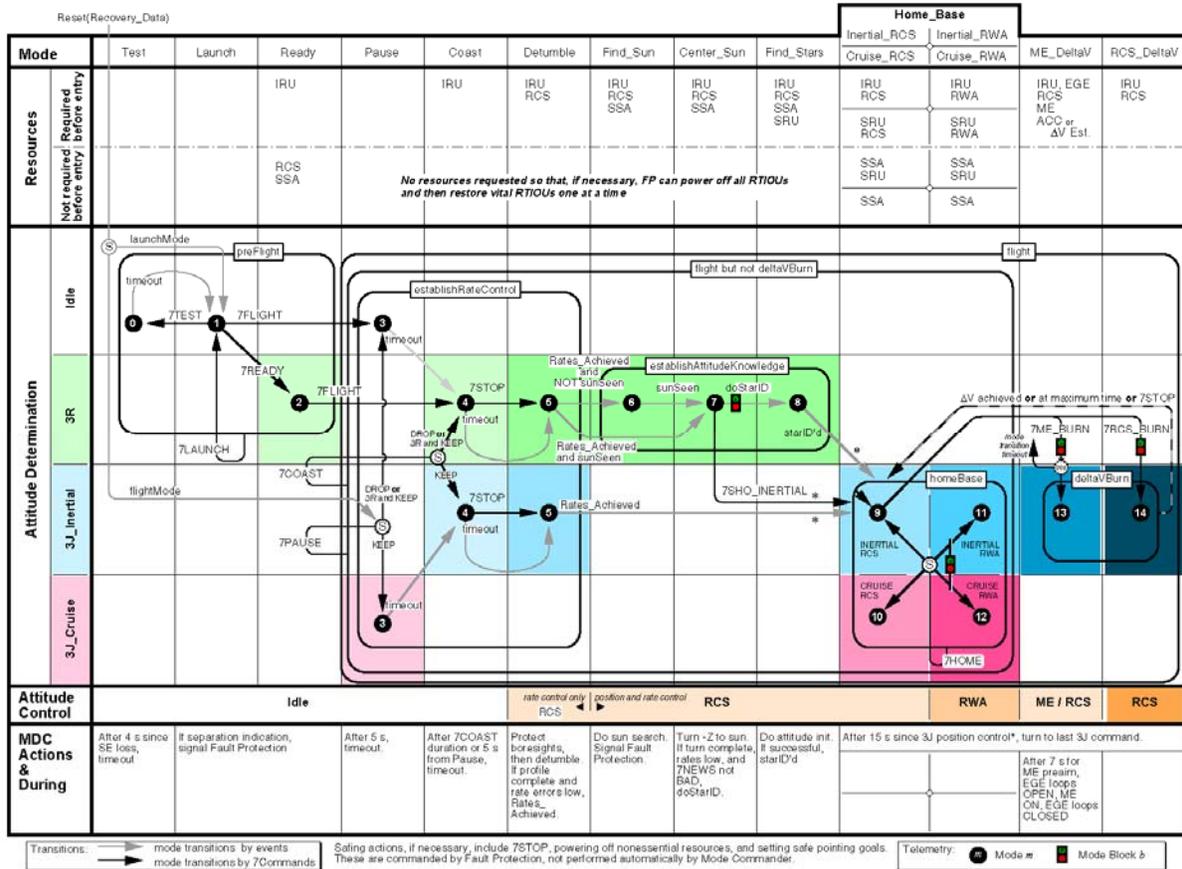


Figure 11. FSW Mode Transition Diagram. FSW states and modes can be traced by identifying needed resources, modes of attitude determination and control, and mode commander actions.

A. Software Control

The state and mode driven software architecture and design have made the FSW quite manageable and quantifiable. Cassini software personnel created an ingenious representation of the software that establishes its execution state throughout the life of the Cassini mission. Software attitude determination and control can only be in one mode at a time, which establishes clear and quantifiable pathways. Each mode enables a unique subset of functional capabilities. Figure 11 expresses the software state and mode driven architecture and the resources required for support.

A mode dictates the types of goals that ACS will attempt to achieve and the hardware resources that are required to meet these goals. Many AACS mode transitions occur autonomously (driven by events). Others are commanded. Autonomous mode transitions always drive ACS toward re-establishment of the Home Base Mode. Within Home Base Mode, ACS provides the operator with two independent degrees of freedom. These sub-modes allow the operator to select a sensor complement for attitude determination (“Celestial-Inertial” using the SRU + IRU or “Celestial-Cruise” using the SRU alone) and an actuator complement for attitude control (RCS or RWAs). A mode transition will not occur until all resources *above* the dashed line which are not already required in the present mode are ready. The flight software does not wait for resources already required in the present mode, or for resources below the dashed line. The software is tolerant of temporary operation without resources below the dashed line.

Except for fault responses, mode transitions are the only case where flight software automatically requests resources in order to honor a command. Commands to remove resources required by a mode are rejected. If a fault

causes loss of a required resource *below* the dashed line, Fault Protection will often try to resolve the problem without changing modes. However, if a fault causes loss of a required resource *above* the dashed line, Fault Protection will generally command a mode that does not require the affected resource.

The central part of the AACCS mode transition diagram is a grid showing combinations of attitude determination and control options. Attitude determination options progress from top to bottom, and attitude control options progress from left to right. Many of the actions performed by the Mode Commander are indicated by the diagram. These include requesting and reserving resources, and directing the attitude determination and control functions. The bottom row of the AACCS mode transition diagram gives information on other important actions initiated by the Mode Commander and on tasks the Mode Commander performs in each mode's specific "During." This term is used to refer to activities performed cyclically (generally once per rate group) during a mode's operation.

Attitude determination is in Idle until Ready mode where it switches to Inertial Relative (3R). An initial inertial reference frame is created and relative attitude is subsequently propagated from gyro data. After star tracker data becomes available, attitude determination switches to Celestial (3J) Inertial where a change to J2000 provides an absolute reference frame. Attitude is propagated from gyro and star tracker data. From there, attitude determination may be switched to Celestial (3J) Cruise which requires only the star tracker.

Attitude control is in Idle until Detumble mode where it switches to RCS control, using thrusters. This is used for most activities until ΔV or finer control is required. RCS ΔV is provided for smaller burns; ME ΔV for larger burns. RWAs are used for precise control during many science observations.

B. Fault Detection, Isolation, and Recovery

Some call it Failure Detection, Analysis, and Recovery, to others it is known as Fault Protection. Regardless of the connotations, its one major goal is to protect the spacecraft, establish telecommunication to Earth, and save the mission. Cassini approached this notion with autonomy. Mission design and requirements imposed a number of constraints on the spacecraft. At Saturn, there is an approximate 1.5 hour one way light time constraint for data to reach, or be transmitted back from, Cassini to the ground. Ground track support is not available twenty-four hours of the day. There are long periods of off-Earth pointing for science observation and collection. One of the most important requirements is the spacecraft's ability to recover and resume from a fault scenario during a critical event. These are but a few examples of why autonomous control capability needs to be very robust in FSW. This

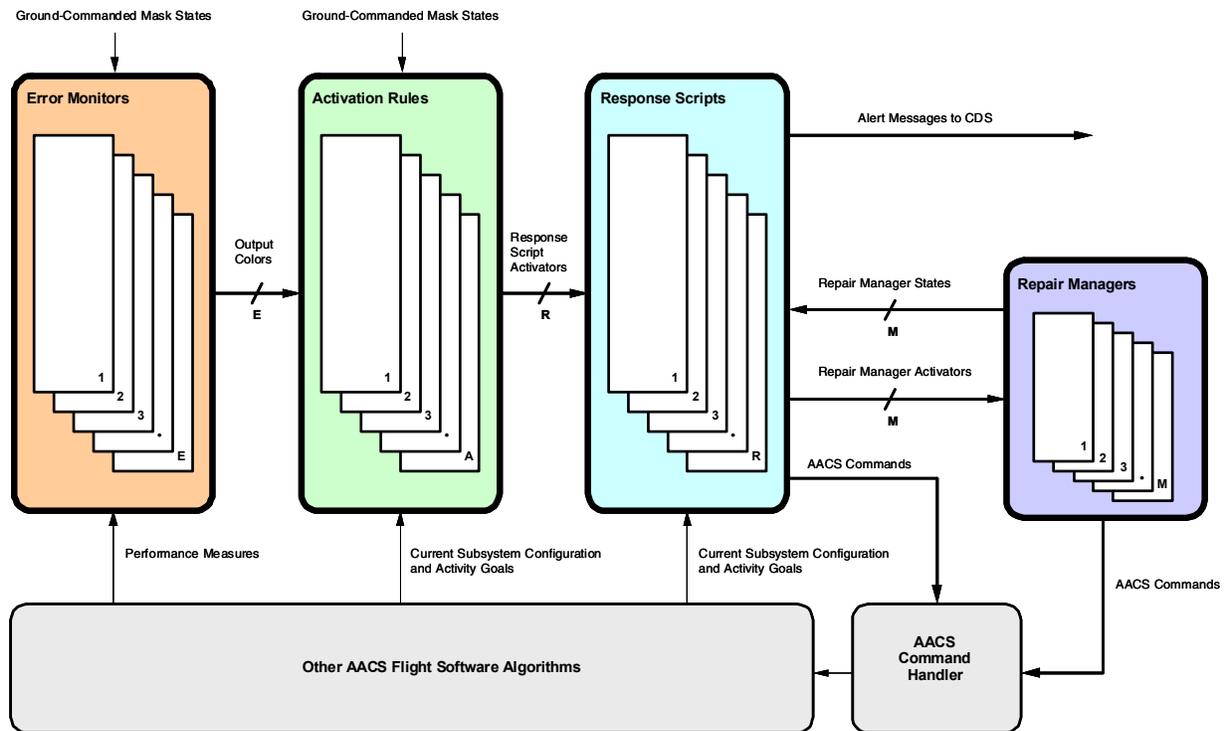


Figure 12. FSW Fault Protection Architecture Diagram.¹² Key components of the FP algorithms to identify Error Monitors, Activation Rules, Response Scripts, and Repair Managers.

establishes two fault tolerant objectives: Fail safe and fail operational.

Fail safe is the term used to insure the spacecraft is safe after a fault and to maintain its safety for up to two weeks without ground intervention. There are several key attributes associated with fail safe: The FSW shall autonomously identify and isolate failed or faulty equipment. Safing-critical equipment shall be replaced. A thermally safe and commandable attitude shall be established. Once safe, the FSW shall wait for further ground instructions. FSW has the ability of keeping the spacecraft thermally safe for an indefinite period by pointing the HGA to the Sun using the two-axis Sun sensor. With the aid of System Fault Protection, a HGA Safing algorithm will command the spacecraft to Earth point after an hour of sustained conditions of no additional faults.

Fail operational is the term used when mission sequences must continue in the event of a fault during time-critical events. Events of Launch, Sufficiently High Orbit (if the launch vehicle failed, achieve a high enough orbit for disposal), SOI, and Probe Relay were all critical events which fall into this category. However, SOI was the key motivation in adding additional autonomy beyond FSW's fail safe objectives. Key attributes associated with fail operational are: The FSW shall autonomously identify and isolate failed or faulty equipment. Equipment necessary to complete time-critical activities shall be replaced. There shall be the ability to handle general processor resets, and ability to rollback and resume the onboard time-critical command sequence. If the Main Engine fails, and the spacecraft fails to achieve orbit during SOI, the prime mission is over. In that regard, the autonomous continuation of a Main Engine burn has to proceed after a quick and accurate fault diagnosis. FSW has the ability to restart the burn if terminated for any reason. Fail operational is dependent on ground-commanded critical command sequences. Upon detection of a fault, CDS will suspend a critical sequence and restart at the last achieved checkpoint. Checkpoints are designated throughout the sequence and insure all goals are met before continuing on with the critical sequence.

After the establishment of control algorithms, fault algorithms were integrated into FSW. Figure 12 reveals there are four primary architecture components to the FSW's fault protection design:¹² Error monitors, activation rules, Response scripts, and Repair managers.

Error monitors test local performance measures against expectations, apply discriminating filters, and then output color-coded opinions. Opinions range from: No opinion (color black), expected performance (color green), unexpected performance which does not merit autonomous response (color yellow), and anomalous performance which merits immediate autonomous response (color red). Opinion color generation is dependent on evaluation of thresholds, and persistence & duration limits during each computation cycle. Error monitor color output results are based on FSW generated performance measurements or ground-commanded tailored mask states. Masking a monitor will prevent autonomous responses by reporting a no opinion (color black). There are 317 different error monitors implemented in FSW.

Activation rules evaluate subsets of the color-coded error monitor outputs dependent of the conditions of current ACS hardware configurations or goal activities. Rules diagnose the most likely cause of anomalous behavior and activate one or more appropriate response scripts. Activation rules are dependent on the output colors of the error monitors, current subsystem configuration and goal activities, or ground-commanded tailored mask states. Masking a rule will prevent autonomous responses to a diagnosis. There are 320 different activation rules implemented in FSW.

Response scripts isolate faulty or failed equipment and respond to recover to a desired level of subsystem functionality. Response scripts are dependent on response script activators from the activation rules, states of the repair managers, or current subsystem configuration and goal activities. Recovery attempts are performed by issuing commands directly to the command handler, directing the activities of the autonomous repair managers, and/or requesting external assistance from CDS-collected alert messages. There are 221 different fault responses implemented in FSW.

Repair managers track the success or failure of past corrective measures for ACS equipment. Managers determine the most appropriate corrective action to take. Repair managers are dependent on action requests from response scripts. Manger actions are exercised by issuing commands to the subsystem and can be thought of as specialized response script subroutines.

Error monitors are integrated throughout the FSW in routines that can establish the earliest test point. This distribution allows test performance and opinion generation to occur anywhere throughout each computation cycle (RTI). The other three FP components; activation rules, response scripts, and repair managers are all centralized and are executed at the end of each computation cycle. Response scripts are prioritized, in the event of two or more response scripts simultaneously being active; the higher priority script takes precedence and can delay the lower priority scripts.

FP architecture is designed to handle single fault and unrelated double fault scenarios. Combinations of monitors can handle multiple fault scenarios that tie redundant hardware to the repair managers which can

autonomously exercise several independent paths of repair allowed by hardware and software constraints i.e., independent prime hardware sets selection, independent bus selections, and composite (mixed-mode) prime sets. Other notable FP management attributes are the resilience to tolerance of: Single Event Upsets (SEU), failure of other subsystems such as CDS, power interruption, corruption or loss of status information, and operator error. In-depth details of these tolerances are referenced in previous papers.¹²⁻¹³

C. Software Objects

Identification and separation of FSW into distinct and unambiguous objects created program set reliability, maintainability, adaptability, and to an extent – reusability. The object oriented approach reduces coupling and increases cohesion. Objects are relatively independent and can therefore be more easily developed, verified, and modified as a result of a change in requirements. Interfaces to the operating environment and peripheral devices are also localized and minor changes can be accommodated in isolated objects. Changes to algorithm equations and their parameters are also isolated to specific objects. Changes to subsystem modes can have broader effects. Table 2 lists the objects which make up the RAM FSW program load.

Missing from the RAM program list is the RAM Verification Block (RVB), which is generally considered part of the FSX object; however, its functions are independent of FSX. The RVB performs initial preparation of the FSW prior to passing control to the RTX and verifies the integrity of FSW execution codes and constants in RAM. It will also perform remapping if necessary, and performs RAM codes/constants patching if so specified in a table in RAM – which allows patch table updates described in Section IX.

Besides the RAM program, ACS FSW is also responsible for the PROM program. The PROM control (Startup ROM – SUROM) program was written in 1750A assembly language and burned into PROM chips. SUROM is tasked to configure and load RAM, accept commands from CDS to load memory and to send memory load request to CDS. Interface responsibilities include generation of heartbeat data, state table data, recovery data, ancillary data, and telemetry to send to CDS. The first program run after any AFC power up or reset is the SUROM to initialize AFC hardware and waits for commands from CDS to perform memory loads. If CDS synchronization signals are not present, the SUROM has a self-AFC startup capability.

D. Software Glue

ACS FSW is designed with five task groups to control and execute software. An interrupt task is the highest priority task, followed by the RTI (timer task), foreground task, background task, and the lowest priority Error Detection And Correction (EDAC) task. The Flight Software Executive (FSX) initiates foreground and background task execution activities, services all interrupts, provides timing and memory read/write services to other objects, and initiates periodic memory scrubs.

The foreground task is the main Ada task which invokes all of the FSW objects in a prescribed order and on average takes up 60% of the CPU usage. When foreground processing suspends, FSX starts or resumes background tasks. The background task is allowed to run until the previously delayed foreground task is scheduled to resume.

Table 2. Final RAM FSW Modules

| | |
|-----|--|
| ACC | Accelerometer Manager |
| ACL | Attitude Controller |
| ACM | Attitude Commander |
| ATE | Attitude Estimator |
| BAM | BAIL Manager |
| CFG | Configuration Manager |
| CMD | Command Handler |
| CMT | Constraint Monitor |
| EGA | Engine Gimbal Actuator Manager |
| FPA | Fault Protect - Analyzer |
| FPR | Fault Protect - Recovery |
| FSX | Flight Software Executive |
| GBL | Global Types and Utilities |
| GBM | Global Hardware Manager |
| IOU | AACS Bus Manager |
| IRU | Inertial Reference Unit (Gyro) Manager |
| IVP | Inertial Vector Propagator |
| MDC | Mode Commander |
| PMS | Propulsion Module Subsystem Manager |
| RWA | Reaction Wheel Assembly Manager |
| SID | Star Identification |
| SRU | Stellar Reference Unit Manager |
| SSA | Sun Sensor Assembly Manager |
| TLM | Telemetry Manager |
| XBA | CDS Bus Manager |

This orderly and deterministic processing is interrupted only by fault conditions. The background task group includes IVP, ATE, CMT, SID, and EDAC background tasks, and on average may take up 4% CPU usage not including EDAC (a measure of idle time.) EDAC averages 30% CPU usage. The RTI task to set the rate group takes less than 1% CPU usage. The interrupt group BCIU, PIU, and 1750A timers average 6% CPU usage. The Ada runtime environment provides an interlock mechanism to control asynchronous access of state data by multiple tasks, and provides subprograms for suspending the foreground task while waiting for timed events or interrupts.

FSX's primary job is to provide timekeeping and scheduling services. FSX starts an infinite loop which establishes the 8 Hz (0.125 second) rate group. There is only one rate group for the FSW. Besides the FSW internal clock driven off of a 1750A timer, FSX maintains both an AACS clock and a RTI clock. The RTI clock is a low cost 32-bit clock that is incremented (ticks) once every RTI. AACS time is synchronized with spacecraft time provided by CDS. FSX is also responsible for making time corrections, handling cycle slips, rate group overruns, and controlling RTI timing jitter.

E. Tool Sets

1. Flight software testbeds

A key factor throughout the FSW development process is the validation of FSW before delivery. The Flight Software Development System (FSDS) was realized as a necessity for FSW throughout the development and implementation cycles of the mission. It was deemed applicable to have a fully emulated high-fidelity ACS subsystem testbed that could be developed in parallel to ACS FSW to provide validation of changing FSW capabilities throughout the entire mission. FSDS implementation started in early 1993 and the testbed was operational in late 1994.

The key to FSDS is that it was created to support the validation of FSW in the ACS subsystem environment. Features were added to FSDS to closely emulate the attitude control subsystem. The redundant ACS subsystem, with dual-redundant data buses and dual AFCs, can run prime and backup FSW simultaneously. FSDS accurately modeled the redundant architecture as illustrated in Figure 13. Millisecond and microsecond level hardware behavior was simulated to create a virtual real-time execution environment. This keeps the correlation of events that happen in real-time, but executes faster due to the host workstation processor. The order of events would occur at the same time as a corresponding real-time testbed. This feature is vital to the realism of actual spacecraft performance. FSW must synchronize to the simulator, and duration of activities must be modeled correctly.

The benefit of this environment is that ACS FSW was required to be in the loop, which gave FSW developers the fidelity needed to test in a pseudo-realistic atmosphere that was comparable to the actual spacecraft environment with respect to ACS concerns. FSDS was found to be the workhorse for functionality testing.

Besides having a strictly software emulated test environment (softsim), the Flight Software Test Bed (FSTB) was developed during the same time period to have FSW exposed to real-time behaviors and

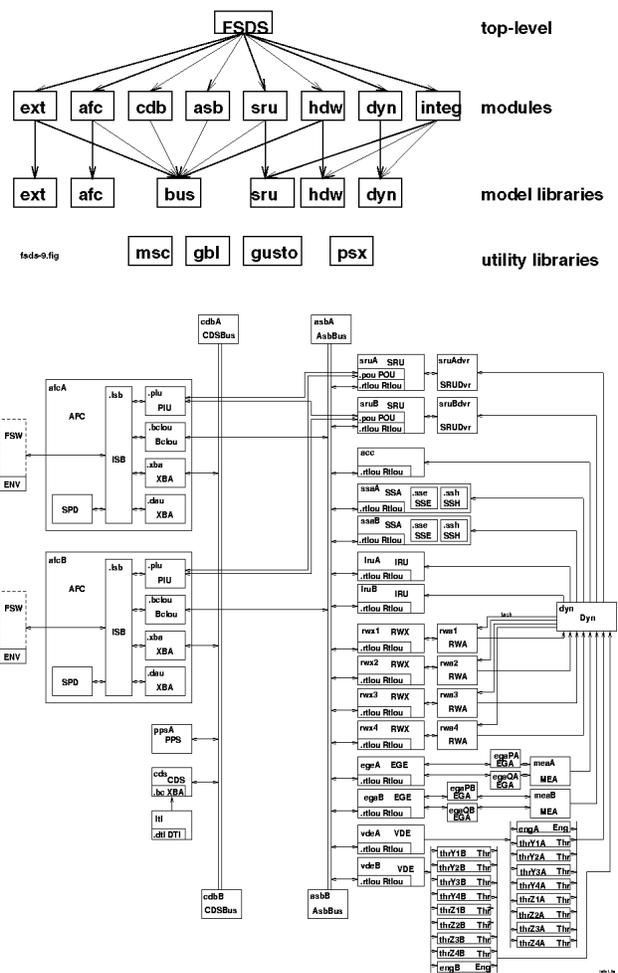


Figure 13. Detailed FSDS Object Oriented Architecture and Model Diagram.¹⁵ Coded in C with layered organization where top-level initializes all modules, modules initialize and interconnect models, and model libraries supply methods and utility functions. The architecture model focuses on the actual hardware configuration.

environment using a COTS processor and AFC board during development. Actual flight processors were very expensive and would have to be shared with CDS FSW development. The FSTB was a valuable real-time equivalent test environment during the pre-launch phase.

For the operations phase, FSTB was retired due to hardware maintenance and expense issues. However, real-time testing fidelity was not in jeopardy. Cassini has two high-fidelity hardware-in-the-loop testbeds for software and hardware integration:¹⁴ Cassini AACS Test Station (CATS) and the Integrated Test Laboratory (ITL).

These embedded testbeds run in real-time and require timely setup procedures and human resources to utilize. The ACS FSW team had to be able to test and verify algorithms and functionality quickly and not be constrained to a few tests per day. At times, there were a dozen FSW engineers developing code, and each requiring a valid test environment. Having a software-based closed-loop softsim running on a desktop workstation that could run faster than real-time at any time with no supervision was a key to timely and accurate testing with little of the overhead associated with the maintenance of keeping up real hardware models. Testbed architecture and operations benefits have been previously published.¹⁵

2. FSC Database

There was a well-defined process for changing flight software starting with the A2 control-algorithm capability FSW builds. Figure 14 describes the process. This process started in 1996 to track FSW relevant issues which were discovered through ATLO or system-level testing. The Flight Software Change (FSC) report database became the FSW team's tracking mechanism for issues to be fixed, not fixed, or deferred to be fixed in future builds. To this day, the FSC process has been followed and FSCs are tracked and reported in test reports, reviews, and release documents. The FSC database is a subsystem-level tracking tool, where the ECR database is considered a system-level tracking tool.

Figure 15 plots a graphical history of FSCs that were opened and closed since 1996. The freeze for launch was around two and a half months before launch day. Table 3 isolates FSCs tracking to FSW build versions. Presently, there are 1523 FSCs in the database. 1437 FSCs were addressed during FSW development. After initial investigations, 86 FSCs were identified as not applicable to address in a FSW build change. The pre-launch development timeframe deferred 167 FSCs to post-launch FSW builds to address future mission specific events or development. See Sections VIII–IX for post-launch FSW development details. For A8 FSW builds, seven FSCs remain open: Five FSCs are associated with active

Flight Software Change (FSC) Process

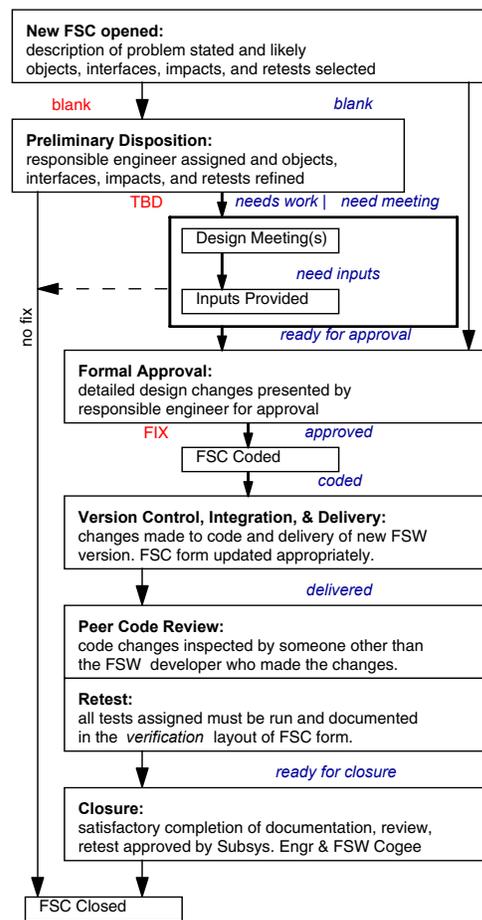


Figure 14. FSC Database Process to change FSW. Key status inputs are: Red – represents the FSW status, blue – represents the FSC document status.

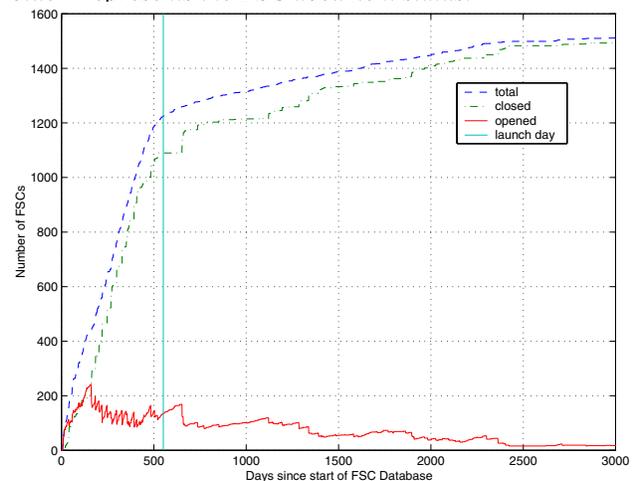


Figure 15. Open/Closed FSC History. Starts from first FSC implementation to show FSC progression of pre and post launch FSW changes.

Table 3. FSC Breakdown by FSW Build

parameter patches that could be implemented in FSW, if a full FSW upload is deemed appropriate by the project, to update corresponding default parameters. One FSC is left open for monitoring in case of a hardware failure, and the remaining open FSC is a RTX operating system bug which is left open for inspection verification after each FSW build release (there are no plans to have TLD fix this issue.)

This database has become a vital metric tool for ACS.

| FSW Version | Capability | FSCs Opened | FSCs Closed | Fix | No Fix | Open |
|--------------------------|--|-------------|-------------|-----|--------|------|
| A4 | Fault Protection and Constraint Monitoring | 187 | 187 | 179 | 8 | - |
| A5 | Full Functionality | 625 | 625 | 608 | 17 | - |
| A6 | Launch, Inner Cruise | 168 | 168 | 167 | 1 | - |
| Pre-Launch Build Totals | | 980 | 980 | 954 | 26 | - |
| A7 | Outer Cruise | 253 | 253 | 242 | 11 | - |
| A8 | Critical Sequences, Prime & Extended Mission Support | 204 | 197 | 144 | 53 | 7 |
| Post-Launch Build Totals | | 457 | 450 | 386 | 64 | 7 |

F. Software Collaboration

Early collaboration between guidance and control analysts and software engineers established a solid well-defined foundation to develop FSW. In the early stages of defining and designing the mission, people had their established roles and software specialists were in the minority. Cassini could have taken the road of inheriting development strategies and approaches from past projects, but there were too many lessons learned that could not be ignored. Architects pushed for changes and new ideas were introduced early on to the entire Cassini team. Rather than the serialization of past development efforts, objectification was the breakthrough needed to establish how software and Cassini did its job. During the software prototyping process, a well-defined coordination between analysts and software was established.

A key factor was early software involvement before algorithm deliveries.⁵ A software unit (module) engineer was appointed months before analyst algorithm delivery. Once the control analyst drafts the algorithm delivery, the software unit engineer presents the algorithm in review. This establishes that the software engineer has the knowledge, understanding, and ownership of the algorithm before applying coding techniques. A key individual was appointed to be responsible for the software-analysis interface. Co-location of FSW, analysts, and subsystem groups was also a key to minimizing interpretation errors and enhancing collaboration within the subsystem.

VIII. Post-launch Phase FSW Development, Implementation and Test Methodology

FSW development and testing spanned another seven years for post-launch development to address outer planetary cruise, and critical sequence support for SOI and the Huygens Probe Release and Relay activities which occurred during the 2004 to 2005 timeframe. After achieving orbit, another four years of FSW updates were performed to address prime and extended mission sustainment. The sustainment effort is still in progress and is currently planned to extend to 2010. Throughout the post-launch phase, the same development and test strategies are used to insure compatible software designs and methodologies. Variations on these themes (focused on reviews and testing) are explained below.

A. Post-launch FSW Development Methodology

It became apparent after launch that the planned deferred FSW development would be separated into two FSW build versions. One to support the spacecraft cruise phase of the mission, and the other to support the execution of critical event sequences.

- 1) Builds A7 (Outer Cruise): These builds contained FSW fixes and updates to support Jupiter campaign activities and the remaining four-year outer planetary cruise to Saturn. FSW version A7.7.6 was the first in-flight full image uplink of software. New capabilities were incorporated to help monitor and detect leaks¹⁶ for the main control mechanism during cruise – RCS thrusters. While attitude control using RWA capability was developed prior to launch, before project management decided to enable the RWA control functionality for science observations during the Jupiter flyby campaign, additional features were added to the RWA control capability to allow for faster recovery if certain in-flight anomalies were to occur. (Several system-level testbed anomalies resulted in additional quick-recovery algorithms for rate, acceleration, and torque limiters to the RWA attitude and wheel rate controllers.) To support the overall mission, the SID algorithm was enhanced to handle extended bodies. To support SOI fault recovery, a deluxe Attitude Initialization capability was added. To support probe relay and tracking, an inertial vector propagation rotating coordinates functionality was added.

- 2) Builds A8 (Critical Event Sequences and Prime Mission Support): A8 was divided into several planned full image version uploads. A8.6.7 supported Saturn Orbit Insertion, A8.7.1 supported Probe Release and Probe Relay. During the prime mission, the project management took a more conservative approach to updating software. Full image uploads and resets to the AFCs took operational hits to science and other subsystems on the spacecraft, and the AFCs and FSW architecture had various work-arounds to update FSW. Therefore, A8.7.2 and its variants, which support the prime and extended missions, are patch loads. Further details are explained in Section IX.

Post launch reviews were held to monitor if FSW development adhered to the appropriate standards and that the proper testing methods were followed. FSW development and testing included fixes to issues encountered pre-launch and deferred, parameter fixes and inconsistencies encountered during flight, deferred capabilities, new capabilities, and capabilities delivered as part of the launch load but were not completely tested or were known to not function properly. An example of a major capability that was addressed post-launch: A robust Main Engine control functionality developed for SOI – which resulted in an energy burn algorithm.

The deferred FSW development and test programs were targeted for two major post-launch reviews. The SOI Critical Events Readiness Review (CERR) was held in April 2004. The FSW was frozen nine months prior to uplink to provide a steady base for critical sequence development and regression testing. Extensive testing did reveal an error in the fault protection burn restart logic and a redelivery was made. Several papers have been written on SOI and the efforts in FSW development and testing.¹⁷⁻¹⁸ SOI successfully occurred on 30 June 2004.

The Probe Relay CERR was held in October 2004. The FSW was frozen eight months prior to uplink, however due to a change in the navigation reference trajectory; pointing errors for the Huygens Probe Relay tracking increased and were beyond the tolerance for probe data relay success if FSW, in case of an AFC reset, used the previous default trajectory data. Therefore, FSW default attitude vectors supporting Probe Relay had to be updated to support the new reference trajectory design. The vectors were changed and the final Probe Relay build was frozen four months prior to upload. Even with the resultant compressed schedule, the full suite of testing to verify and validate FSW for the critical event sequence was achieved. The Huygens Probe Relay with Cassini was successfully accomplished on 14 January 2005.¹⁹

After the execution of the in-flight critical sequences, the prime mission was under way. In order to support prime mission events such as low-altitude Titan flybys, Monopropellant Tank Assembly recharge affecting thruster magnitudes, and orbit trajectory changes, FSW had to be updated to support the effects of these events. A symbiosis between FSW and operations teams was achieved by updating necessary FSW parameters in flight and finding ground operation work-arounds for software issues that were not deemed essential for an update.

B. Post-launch FSW Test Methodology

The test methodology for post-launch activities followed the pre-launch methodology. The method consisted of incrementally increasing complexity and functionality through the phase of integration testing for each build, which culminated with scenario and functional testing of the complete set of software. To support the post-launch stage of the mission, additional testing had to be considered for full and partial FSW uplinks and contingency planning. Testing and validation of these activities became critical additions and major foci for the operations testing philosophy.

For A7 builds, regression, scenario, and functional testing were performed to verify and validate changes to FSW to support the interplanetary outer cruise to Saturn. New FSW functionality was unit/FSC tested. FSW went through subsystem and system level testing before being uplinked to the spacecraft. Uplink Readiness Reviews (URR) were the major milestones to approve FSW for uplink. The FSW development status, FSW testing (both on the subsystem and system level), FSW test reports, and uplink procedures were presented to review boards.

For A8 builds, the same A7 build test philosophy was followed. An abundance of additional testing was focused on the critical events of SOI and Probe Relay.^{10,17-19} To support the prime mission, parameter and uplink procedure testing continued under the same test philosophy. During the prime mission, key science objectives were tested extensively which included post-event reconstruction i.e., Titan low-altitude flybys,²⁰ and low-altitude Enceladus flybys.

The ACS operations group has anticipated and evaluated potential problems and determined mitigation strategies where appropriate. Environmental circumstances established the need to evaluate Safing attitude changes due to orbit inclinations, and potential ACS hardware failure scenarios. Appropriate measures were taken to keep the spacecraft as safe as possible and the only way to verify scenarios was through testing. All of the strategies were tested in subsystem and system level test environments.

Table 4. ACS FSW Upload History since Launch (15 October 1997)

| FSW Version | Upload* Classification | Capability | Key Update Description | Upload Date |
|-------------|------------------------|--------------------------------|---|-------------------|
| A6.5.7 | Patch/SSR/BAIL (A) | Venus Cruise | <ul style="list-style-type: none"> A6.4.15 (launch load) on default SSR partition, A6.5.6 (cruise load) on non-default SSR partition IVP, S/C time update, AFC swap | October 30, 1997 |
| A6.5.8 | Patch/SSR/BAIL (A) | Earth Cruise & SRU-B alignment | <ul style="list-style-type: none"> IVP, S/C time update Fix Backup AFC telemetry collection period, changed default Accelerometer nominal drift, adjusted SRU-B alignment quaternion (due to 3/4 mrad misalignment) | May 1, 1998 |
| A7.7.6 | Full Load (B) | Jupiter Cruise | <ul style="list-style-type: none"> IVP rotating coordinates & conical radar scan capabilities SID extendend bodies, deluxe attitude initialization capabilities RWA celestial only attitude estimation | March 7, 2000 |
| A7.7.7 | Patch/SSR/BAIL (B) | Parameter & BAIL update | <ul style="list-style-type: none"> IRU, IVP, RWA, and default thruster magnitude updates BAIL FSW A6.3.B patch (mass properties, Cassini-Sun vector, cruise updates) | April 13, 2001 |
| A8.6.5 | Full Load (A) | Energy Burn | <ul style="list-style-type: none"> In-flight proof-of-concept for new Energy burn algorithm performing TCM-19b | February 16, 2003 |
| A8.6.7 | Full Load (B) | SOI | <ul style="list-style-type: none"> Parameter updates to support SOI FSW/FP updates for SOI | April 27, 2004 |
| A8.7.1 | Full Load (A) | Probe Release & Tracking | <ul style="list-style-type: none"> Parameter updates to support Probe Release and Relay Reference trajectory updates for Probe Relay | October 2, 2004 |
| A8.7.2 | Patch (A) | Titan flyby & Tour Telemetry | <ul style="list-style-type: none"> Finer telemetry resolution to external torque for Titan flyby density reconstruction Finer telemetry resolution for Delta-V telemetry Detumble acceleration Z-axis update | May 26, 2005 |
| A8.7.4 | Patch (A) | MTA-Recharge | <ul style="list-style-type: none"> Post-MTA recharge default thruster magnitude updates | April 11, 2006 |
| A8.7.5 | Patch/SSR (A) | 2007 Safing Attitude | <ul style="list-style-type: none"> Default secondary Safing vector pair update Default thruster magnitudes for 2007 tour updates IRU-A scale factor error updates RWA phantom momentum FP tier count change | January 30, 2007 |
| A8.7.6 | Patch/SSR (A) | 2008 Safing Attitude | <ul style="list-style-type: none"> Default secondary Safing vector pair update Default thruster magnitudes for 2008 – mid2009 tour updates | January 8, 2008 |

- * - Full Load = Full FSW image loaded to SSRs with Prime/Backup AFC resets
 - Patch = AFC patch-table patch, Active RAM patches, Backup AFC reset
 - Patch/SSR = Patch + SSR ALF patch (parameters survive load from SSR)
 - Patch/SSR/BAIL = Patch + SSR ALF + BAIL patch (parameters survive load from SSR)
 - (A or B) = Which AFC (A or B) is Prime after the update

IX. Flight Software Operations

The mission operations team has to address FSW practically every single moment of every single day. FSW interacts with every facet of the spacecraft which in turn affects ground operations support. Moments after launch, ACS FSW telemetry and the FP event log were monitored and updates to FSW were planned and performed. Eleven years later, the same exact concerns besiege an ever-changing/high-turnover rate operations team. Established and/or the establishment of processes and procedures are key in retaining the history, knowledge, and know-how of FSW operations.

A. Key In-flight FSW Update Events

The AFC and SSR configurations, having redundant block and multiple partition configurations, provided the opportunity to have multiple versions of FSW loads. When an AFC is *prime* it exhibits full ACS FSW capabilities. A *backup* AFC exhibits minimal software functionality and maintains telemetered status. During critical events the backup AFC will become a hot backup which allows the backup AFCs to continue processes of the prime AFC if a fault anomaly results in an AFC swap. At launch, the prime and backup AFCs contained the launch FSW load which was inhibited from performing maneuvers using thrusters or actuators, but actuators were allowed to have some minimum parking abilities.

One day after launch, via ground commands, the backup AFC (AFC-B) was loaded from the non-default partition of the SSR with the Cruise FSW version (A6.5.6). After a brief checkout, the AFCs were swapped and the new backup AFC (AFC-A) was updated with the Cruise load. The one-day turnaround of updating FSW without a full image in-flight upload could only be effortlessly accomplished with the SSR partition concept, which is explained in detail in the next section.

Fourteen days later the patch table capability was first used in flight. Patches containing A6.5.7 FSW patch table updates were uplinked to update the SSRs, and the AFCs were reloaded and swapped. AFC-A was designated the prime AFC. Table 4 identifies all of the in-flight changes to FSW from launch to the end of prime mission.

The first full in-flight software upload was FSW version A7.7.6, which had the second highest total of software fixes. Factors that contributed to the amount of fixes were keeping a core software development team from the pre-launch team to address a majority of the deferred software FSCs. The load addressed several key functions needed to support future mission objectives.

FSW version A8.6.5 addressed a first-time JPL event demonstration of an energy algorithm to perform a burn cutoff during TCM-19b on 2 October 2003. This algorithm was developed for the upcoming critical event of SOI. The algorithm, commands, telemetry and overall concepts and capabilities were in-flight tested, verified, and validated nine months before SOI.

Critical events were performed on two different full in-flight uploads. SOI was performed on FSW version A8.6.7. Probe release and tracking were performed using FSW version A8.7.1. These events were covered extensively in the previous section.

The next four updates were patch table uploads to support prime and extended mission objectives. These FSW builds represent a more conservative approach taken by the project. From an operations point-of-view, the best approach was to minimize disruptions to science observations and effects to other subsystems. From a software point-of-view, the FSW architecture allowed for variations of updates and patches which ideally mimics a full in-flight upload with less system impacts. Also, the sum total of all the parameter updates did not exceed the limitation of the patch table. These update variants and limitations are explained in the next section. The program's conservative approach was to also lessen the possibility of errors that could be introduced after a full FSW upload with extensive functional modifications. If a situation ever occurred, where FSW logic had to be changed, technical justification does overtake programmatic. A major driver on the number of updates needed during this phase was the changing inclination in the orbit trajectory. Practically every year during the prime mission, the default Safing attitude needed to be updated to assure that the spacecraft be in a safe attitude both thermally to science payloads, and physically to protect the star tracker from bright bodies such as the rings or Saturn.

B. Strategies for Updating and Loading FSW

The intentional modular design of FSW provided the architecture to support planned in-flight FSW updates. The form of uploads range from full image to patch table uploads. The type of updates can range from full build changes to default parameter, and active parameter patches. The prime mechanism used to store FSW images on the spacecraft are the two 2.1 gigabit (usable memory) SSRs under the control of CDS. SSRs contain volatile memory and 64 megabits per SSR are reserved to store FSW images.⁹ Both CDS and ACS can store two sets of their programs within the allocated memory partitions. There is a default program set and non-default program set. Default and non-default partitions provide a fail proof method to update FSW.

Ground operators can upload a new version of FSW to the non-default partition of the SSR and then command the AFC to load from the non-default partition. During the load, the AFC will go to ROM, SUROM will initialize the AFC to accept the new load and then transition to RAM. If a checksum problem or other unforeseen run-time problems occur, the AFC will stay or go to ROM. Either ground commanding or autonomous FP response will reload a good version of FSW from the unchanged default partition. The data format used to store FSW images in the SSR are Assisted Load Format (ALF) blocks. An ALF is made up of twenty-two 16 bit words with even/odd word checksum validations.

There are five locations where ACS FSW is stored: Two SSRs, the prime and backup AFCs, and the Backdoor ALF Injection Loader (BAIL). Only the BAIL has a stripped-down version of FSW stored in 16 megabits of non-volatile Electrically Erasable Programmable Read Only Memory (EEPROM), which provides the capability for FSW to command the spacecraft's HGA to Sun point. Power converters in the SSRs and AFCs provide at least 37 millisecond of capacitive power to retain software in volatile memory and therefore can not retain software after an extended power outage. To address possible deep under-voltage conditions, the BAIL functionality was added to assure a thermally safe and commandable attitude during ground-assisted CDS recovery. Without contact from the prime AFC, a thirty minute watchdog timer will expire in the BAIL and it will start transmitting its program contents to both AFCs on both AACCS buses in the hope that these FSW programs can be accepted and loaded by a healthy AFC.⁹

To account for parameter updates during Cassini's lengthy mission, other mechanisms were provided to update FSW. A number of FSW ground parameter commands were added to update RCS attitude controller parameters, attitude estimator parameters, constraint monitor parameters, ME thruster parameters, mass properties, and various FP-based parameters. Parameter commands have been validated during both pre and post launch phases. However, for operations, the command arguments are highly scrutinized before transmission to the spacecraft.

There are two forms of memory write commands. One modifies *logical* addresses and the other modifies *physical* addresses. The logical address command has a constraint to match the FSW version. The physical does not

and is ground-restricted from use in flight. Since the SUROM has the capability of remapping the FSW memory map when loading into RAM, using a physical address to change data is highly susceptible to remapping error. The use of logical addresses eliminates the remapping concern. Unlike established parameter commands, the arguments for memory write commands are highly susceptible to error uncertainty, unless properly handled. The next section addresses this concern.

Parameter and memory write commands affect the RAM program which is considered the *active* parameter set. If a reset occurs or the FSW is reloaded, the changed active parameters will not survive. Due to the changing environmental effects or spacecraft consumables, which need to be accurately represented in software, there are some parameters that need to survive reset/reload conditions to aid in a recovery process. These parameters are considered the *default* parameter set.

To address a permanent change to the default parameter set, the AACS patch table enables FSW patches to be applied across the different load locations (AFCs and SSRs) and if necessary, across different version of FSW. The patch table is located at a fixed logical address common to all ACS FSW versions and is capable of updating up to specified length data words. See Figure 16 for the patch table format. Any change requiring more than the allowable number of words will have to be re-planned to a full FSW image upload. The Ram Verification Block (RVB) checks if anything occupies the addresses of the patch table. If the patch table FSW version ID matches the recipient FSW image, the RVB will apply the patches before activating the RAM program. The RVB contains logic to check a completion flag if the patch table patches have already been applied; it will not attempt to apply the same patches twice. The completion status flag will only be set after the entire content of the patch table is applied. Therefore, if it is interrupted during the patching process, the RVB will recognize that patches still need to be applied at the next AFC reset attempt. The patch table method is a way to safely preload code changes and apply the changes in one complete activity.

Figure 17 shows the concept and patch steps taken to apply a patch table patch to the SSR and the prime AFC. The SSR patch table patch is to address the situation if the AFC needs to take a FSW load from the SSR. For this scenario, the AFC requests a load from the SSR, the full FSW image plus patch table are loaded in the AFC. When the RVB is run during startup, the FSW executive will recognize that a patch table patch is present and the patch will be applied resulting in FSW that reflects the latest FSW version. This process is done to the backup AFC which does not partake in nominal spacecraft modes and therefore does not affect operations. This accomplishes two objectives, to verify the SSR patch and to update the backup AFC to the latest FSW version. If an AFC swap were to ever occur, the operations team would be confident that the latest FSW would be in use during recovery operations.

In order not to affect operations and other subsystems, the prime AFC is not reset nor is there a swapping of AFCs. Active patches are applied to the FSW running in RAM on the prime AFC. The active patches equate to the changes of the default parameters of the latest FSW version, so even though the FSW version still registers as A8.7.1 (an older version of FSW); the RAM FSW is equivalently A8.7.6 (the latest version of FSW). With the patch table applied to the prime AFC, if an AFC reset were to occur, the patch table would be applied during the next startup process and the latest FSW version would be registered in RAM.

C. FSW Parameter Patching Techniques

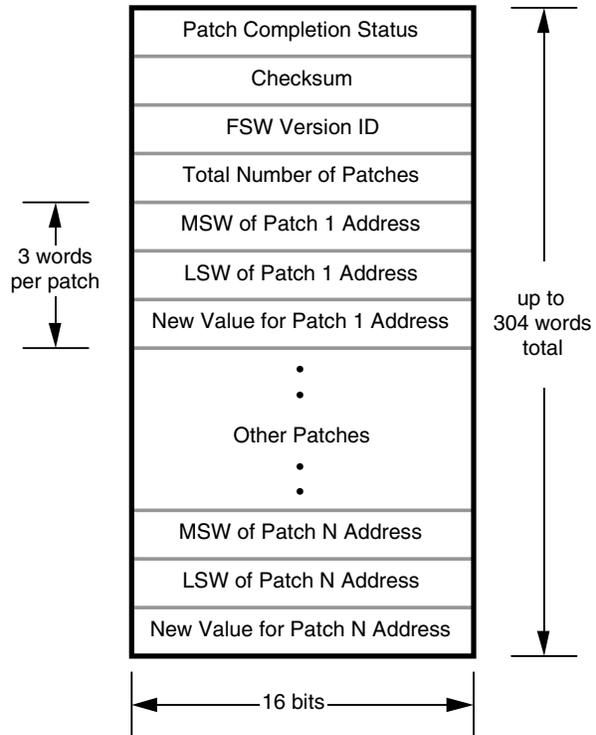


Figure 16. Patch Table Format.⁹ Allows words to be modified through the ACS FSW memory map.

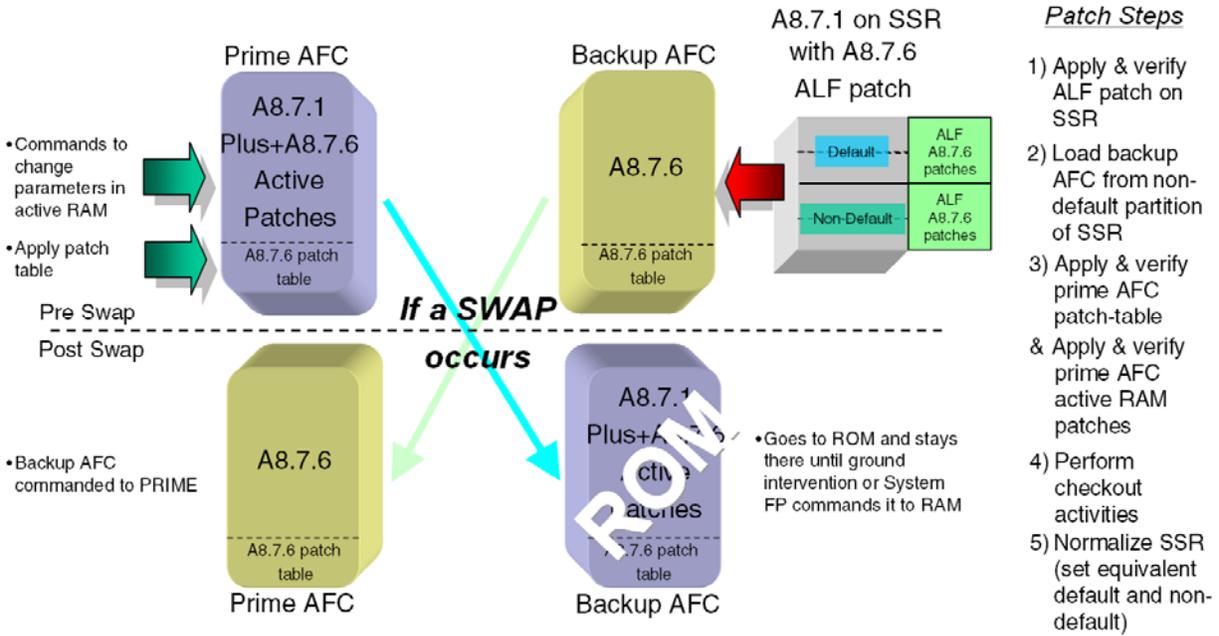


Figure 17. Simplistic representation of a SSR ALF patch and Patch-table patch to update a FSW version without having to reset the Prime AFC. FSW version A8.7.1 was the last full upload. Since then, patch updates have only been applied to update the FSW version. A8.7.6 is the latest patch build to be uploaded in flight.

Patching with memory write commands require user-supplied hexadecimal arguments. There is always an associated risk of software failure when patching FSW. Several missions have experienced software mishaps associated with patching. ACS mitigates risk when patching a FSW parameter by following a strict process of identification, ground verification, and in-flight verification. Any patch that involves software shall be identified by, provided by, and verified by a knowledgeable ACS FSW engineer. These values will be certified by another FSW engineer or FP engineer.

1. Identification/Classification of a patch:

- 1) Identify a parameter as a *variable* or *constant* and determine the current value associated with it. There are established procedures to update variables or constants. Data classifications are explained in the Software Data Definitions of the Lessons Learned Section.
- 2) The current onboard FSW version Memory Map is used to determine *logical* addresses for patching. The FSW version is verified through telemetered data before proceeding with a patch.

2. Thorough ground verification, validation, and test:

- 3) Verification and testing involves using the exact memory write command and patching process used to update the FSW in flight.
- 4) System-level ground testing in ITL is done using the same configuration state as the actual spacecraft.
- 5) Effects of the patch in FSW and the spacecraft environment are verified in both subsystem and system level testbeds.

3. In-flight verification and validation:

- 6) Establish a baseline value in case a patch needs to be undone by checking and verifying the initial state of the parameter on the spacecraft.
- 7) Verify that the MROs match the expected pre-patched value state – if it is not, something is wrong i.e., the wrong address was patched.
- 8) Perform the patch using ground-validated commands.
- 9) Perform post-patch verification for correctness.

All MROs are performed three times to anticipate any loss of data during telemetry downlink. MROs are captured in both real-time and record telemetry. Every active parameter patch is performed using this exact process. Established ground parameter commands, mentioned in the previous section, can utilize steps 2-3 for a verification process if deemed applicable.

D. Design Maintainability

From the beginning of FSW development, the maintainability aspect was always a key goal for the end product. These efforts consisted of:

- 1) *Simplicity* by providing definitions and implementing functions in the most non-complex and understandable manner as possible. E.g., coding, design, flow, software development guidelines, and simple architecture.
- 2) *System Clarity* by providing a clear and understandable description of the program structure. E.g., programming style, coding guidelines and peer reviews, and a Software Specification Document.
- 3) *Modularity* by providing a structure of highly cohesive modules with optimum coupling. E.g., object oriented approach to maximize cohesion, minimize coupling, hide information, and well defined interfaces.
- 4) *Self Descriptiveness* by providing an explanation of function implementation. E.g., comments, identifiers, layered diagrams and Software Specification Document descriptions, well named objects and descriptive names for data and procedures.
- 5) *Exactness* by insuring software code performs desired requirements, and eliminates code not supporting required functionality. E.g., RTMs and optimizing the compiler.

These attributes have contributed to the success of software maintainability in the operations phase, and especially helps when code needs to be analyzed for functionality and/or compared to requirements. This allows non-software engineers to understand the interactions and intent of modules and functions without understanding the coding language.

E. Configuration Maintainability

Using the patch table upload process in the latter part of the mission, results in greater ground responsibility to maintain accurate version control. Patching without full uploads or intentional post-upload AFC resets results in multiple versions of FSW on the spacecraft. The prime AFC has an older version of FSW (however, with active patches it is equivalent to the latest FSW version), the backup AFC has the latest version (loaded from the SSR), the SSRs have patched older versions of the FSW (when loaded to the AFC, the resultant image is the latest FSW version), and BAIL has an even older version of FSW. It becomes apparent that accurate records need to be maintained to identify FSW versions on the spacecraft and the parallel development versions on the ground.

Records take the form of version control and documentation. The version control mechanism used by the FSW team continues to be SCCS. While new JPL projects use more complex versions of configuration management which allow modeling and document requirements tracking techniques, the process established during the Cassini FSW development phase is quite adequate for operations. As long as the process is documented and there is a method to retrieve any version of FSW, keeping it simple contributes to maintainability.

The Release Description Document (RDD) is a required document that describes new functionalities available in FSW as well as the build process. It incorporates all test reports, and identifies all problem reports written and fixed by individual FSW versions. The results of the Software Requirements Certification Review (FSW delivery and certification of flight readiness) are also captured. The RDD is a key document for delivery, integration, test, and identification.

F. Software Procedures

With a reduction of workforce in the operations phase, FSW procedures become a key component of fulfilling the FSW process. Having procedures for generic FSW tasks allows for procedure tailoring, and as long as the tailoring effort is understood, the process is sound. Therefore, the concept for tailoring must be applied to every FSW procedure, which must result in reviews and validation for each procedure. Until recently, each FSW upload entailed different objectives, and if care is not taken, the use of old procedures may not result in the desired actions. This verification and validation technique is applied to all ACS FSW related procedures. ACS FSW has established procedures for:

- 1) Full FSW uploads. These procedures cover the upload of a full FSW image to the SSRs, loading the backup AFC from the SSR and swapping the prime/backup states of the AFCs.
- 2) Patch table parameter updates to address changes to constants and default parameters when an AFC reset occurs.
- 3) SSR ALF patching to capture constants/default parameter changes in FSW that survive a load from the SSR.
- 4) Active RAM parameter patching to address changes to variables. These procedures apply to patches that do not have a requirement to survive an AFC reset condition.

There are system-level procedures that are specific for recovery efforts, which are continually updated to be used in general spacecraft recovery, and these system-level procedures are meant to be used “off-the-shelf.”

G. Testbeds

Post-launch management adopted the same pre-launch philosophy of maintaining the FSDS softsim and hardware-in-the-loop testbeds. This is a needed capability for continuing FSW development. Cassini, with its deferred FSW development, placed high value on maintaining its testbeds. However, this is a key component to the success of any mission. Testbed capabilities need to be developed and added in parallel to FSW to support events during the mission timeline. Major efforts occurred to update FSDS and ITL by implementing a Titan atmospheric torque model to simulate the effects Cassini would encounter during low-altitude flybys.¹⁵ With the anticipation of upcoming events, another major effort was taken to implement an Enceladus plume density model to support and test spacecraft controllability during Enceladus flybys as low as 26 kilometers.

The operations FSW team is considerably smaller than the pre-launch development team. During pre-launch, as many as fourteen developers supported the team. At the end of the prime mission, that number has reduced to two team members. A softsim (FSDS) was invaluable to making a small team very productive at development, test, and problem investigation. Having the capabilities of simulating these stressful environments allows an abundance of testing for nominal and off-nominal scenarios to establish margin envelopes for spacecraft safety. The services of a full system-level ITL was a key factor in finding, understanding, and fixing software anomalies early and on ground-based flight or engineering hardware rather than the spacecraft. Having results from both softsim and hardware-in-the-loop testbeds provide a self-checking pair capability, which is an invaluable resource to have when testing FSW functionality. Testing result differences indicate discrepancies that need to be further studied to determine if there are FSW problems or issues with the testbeds.

H. Regression Testing

Regression tests that were established pre-launch are still part of the regression tests during post-launch FSW verification. Regression tests that no longer apply due to the mission phase have been replaced by tests that capture current mission scenarios. Tests that verify and validate new FSW functionality have been incorporated into the regression test suite. Whether it is a parameter change or functionality change, a core regression test suite is always executed on new FSW builds. This process is used to catch the possibility of undesired effects to the subsystem and system due to FSW changes. Regression testing details have been covered in previous papers.^{10,21}

I. SLOC Comparisons

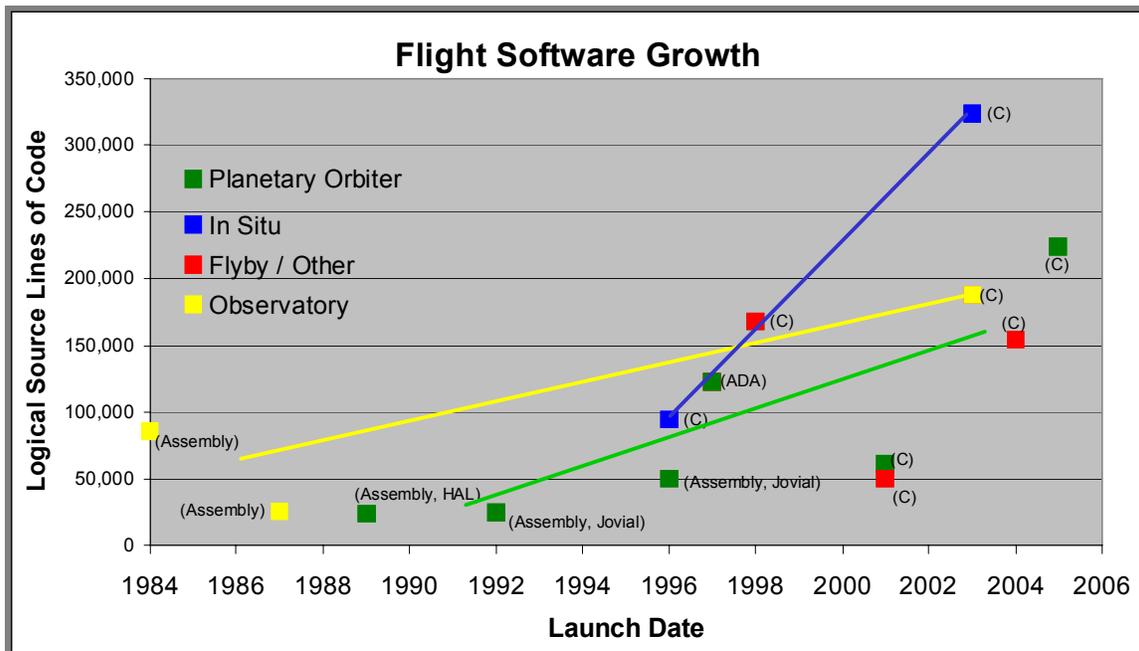


Figure 18. Historical SLOC Comparison of Flown Missions.²² JPL-gathered data on past missions (development language) and how missions have increased in complexity and reliance on software.

In general, Industry measures software complexity, cost, and effort by Source Lines Of Code (SLOC) counts. SLOC metrics are established methods which allow scope comparisons to previous missions, and to an extent provide a means to predict future mission FSW effort and cost. Figure 18 plots logical SLOC counts for fourteen different NASA missions. For planetary orbiters, the trend shows how newer missions are more reliant on software-intensive systems. More autonomy creates more complex software, and among orbiters, Cassini SLOCs almost tripled the previous orbiter missions. The pre-launch ACS FSW logical SLOC (not including CDS FSW) estimate was 48,000 SLOCs. At the end of the prime mission, the ACS FSW logical SLOC estimate was 55,000 SLOCs.

X. Lessons Learned

The following are observations that have been discerned in support of ACS FSW during the mission operations phase.

- 1) Importance of adopting FSW design, development, and testing techniques and evaluating their impacts to the system and subsystem during operations.
- 2) Importance of the operations FSW team to commit to FSW ownership and the responsibilities associated with it, and not to only rely on past software personnel support.
- 3) Importance of honing the techniques of averting FSW risk in mission specific environments.
- 4) Importance of maintaining accurate records for software versions. These methods help to keep corporate memory during longer missions (described in the previous section.)
- 5) Importance of having robust methods to update/patch FSW (described in the previous section.)
- 6) Importance of retaining and maintaining testbeds for FSW development and test during longer missions (described in the previous section.)
- 7) Importance of verifying and validating new FSW capabilities in flight before the functionalities are needed during critical events.
- 8) Importance of understanding impacts of data definition types to memory protection, and importance of legal range checking in flight and ground software.
- 9) Importance of reevaluating basic FSW capabilities and improving key attributes or proposing work-arounds, i.e., ability to retain data across resets, and catching exceptions to good coding techniques.

Details of the lessons learned from these observations are described below:

A. Success is in the Details

This is by far the most important point that one should take away from this paper. The quality and time spent on details during the design and development phases of ACS FSW has provided a software suite that has stood the test of mission objectives. When the responsibility of the spacecraft shifted to operations, the methodology and the practice of in-depth attention to detail continued. Anticipating what may or may not go wrong with upcoming events, or preparations to respond to possible anomalies have continued to make Cassini successful to the end of its prime mission and beyond. Sweating the small stuff is a necessity for operations to minimize complacency and ensure mission success. Much of the merited and rational paranoia was directed at FSW because of its importance to the spacecraft and mission objectives. Attention to detail must happen first before achieving success.

As an example: While procedures were written for software uploads and maintenance prior to launch, the engineers who wrote the procedures usually did not continue into the operations phase. The authors did not see the procedures to fruition. To take procedures at face value without the understanding, know-how, confidence or effect it has in software can lead to failure. Pre-launch processes and procedures were either revalidated or rewritten post-launch to capture details gained from in-flight experience.

No matter which phase the mission is in, the practice of evaluating technical scope, performing trade studies, analyzing risks, monitoring system performance, and handling schedule conflicts must be in the forefront. The project, spacecraft operations office, and ACS group have balanced these attributes and applied them to every type of software change which contributed to the success of the mission.

B. System-level Considerations in Subsystem-level Development

Early considerations of high-level system-level interactions with the ACS subsystem benefited the design and development aspects throughout the mission. Prototyping between subsystems not only consisted of the hardware and software interface designs, but the human elements as well. Group interactions were thought out between analysts, software developers, integrators, and testers. Prototyping resulted in a software life cycle that was unified within the affected groups. Work agreements, schedules, and document templates were established and refined throughout software prototyping. The collaboration between groups took time, but once a consensus was reached,

the work expectancy and commitment was solid. Regular working groups and frequent reviews were instrumental in keeping with development commitments and a unified design.

As examples: Early system evaluation revealed that fault protection had to be considered at the start of the project. Cassini started FP design a year earlier than previous JPL missions. This allowed for a systematic approach to requirements generation, design aspects, and test planning. Early scoping resulted in the establishment of testbed support and development. As a result models were unified between different hardware and software testbeds.

Considering the full scope of the system and its impact to the subsystem resulted in early establishment of self-imposed requirements that contributed to the success of the mission.

C. Software Collaboration and Ownership during Operations

While the development processes remain consistent throughout new and old projects, each project is very different and FSW is the one element that needs to adapt the most to achieve mission goals. Having a software community that shares acquired knowledge of their experiences within the different projects establishes a concept of software collaboration. This also means; once a FSW engineer supports a project, that engineer may be called upon to support it again long after he or she has moved on to newer projects. There should always be a sense of responsibility when dealing with FSW. Because of the Cassini legacy and sense of continual collaboration, the initial analysts, architects, and FSW developers have provided the necessary support. This has provided Cassini with the best available help.

However, due to the length of the mission, the expectation for people to remember nuances cannot be assumed. This realization has stressed the need and importance of taking ownership/responsibility of the FSW during operations. During the pre-launch architecture, design and implementation phase, FSW engineers firmly understood the software and the concept of ownership. If they did not perform their job to satisfy engineering goals, the spacecraft would not launch. The operations FSW team is also subjected to supporting growing scientific goals. Whether objectives are engineering or scientific, the operations team needs to value this same concept. Whether a FSW engineer was assigned to lead initial development, to provide post-launch algorithm support, or to test FSW compatibility with extended mission scenarios – without ownership, the depth of responsibility is not perceived. Delegation grants authority, but not responsibility. During operations, Cassini has been fortunate to retain FSW engineers who took the ownership responsibility and responded accordingly. Software collaboration and responsibility have contributed to the success of the mission.

D. Risk Aversion

During operations, there is a perception that the FSW team becomes reactive rather than proactive. During the pre-launch design phase, new concepts and ground-breaking strategies are welcome; during operations, a conservative approach may suppress creative designs. The perception focuses only on the idea that operations react to anomalies. While some of the operations FSW team's abilities may become dated with certain design techniques, risk aversion techniques become cutting edge. As a result, the operations FSW team is more proactive than any of the past teams. Supporting operations will result in a well-rounded FSW engineer. FSW has already proven itself by successfully guiding Cassini throughout the prime mission. However, the question that still needs to be asked and addressed is: "What else needs to be done to keep the FSW anomaly-free in upcoming mission scenarios?" Cassini ACS has focused on averting risk by trying to prevent the likelihood and adverse consequences of future anomalies, and not just react to problems. This has been accomplished by continually exercising fault scenario tests, evaluating parameter selections, being aware of current mission environments, monitoring current spacecraft health, and establishing FSW work-arounds. While problems may have been kept to a minimum, the tasks of addressing risk aversion are never complete.

E. Major First Time Software Events

If major functions are added to software to perform critical activities, ground testing alone should not be considered adequate. A proof-of-concept should be performed in flight whenever possible. FSW changes to attitude determination and control can usually be verified in flight. FP changes may not be programmatically feasible to test in flight; however, the verification of FP parameter values can still be performed.

FSW version A8.6.5 was uploaded sixteen months before SOI, and a major driver for such an early uplink was to validate a first-time implementation of an energy cutoff algorithm to terminate a ME burn. TCM-19b was designated to be the first maneuver using the new algorithm. The algorithm was an ACS-intensive addition in FSW. The energy cutoff concept was possible due to the object oriented design of the FSW which allowed additional modifications without having to redesign the software architecture. All the hooks necessary for a new energy burn design were available in FSW.

$$\Delta E(t) = \int_0^t \vec{V}_{NoSOI}(\tau)^T \bullet \vec{A}_{SOI}(\tau) \div 10^6 d\tau \quad (1)$$

$$\begin{aligned} \Delta E^{ideal}(t) &= \int_0^t \vec{V}_{NoSOI}(\tau)^T \bullet \vec{A}_{SOI}^{ideal}(\tau) \div 10^6 d\tau \\ &= \int_0^t \vec{V}_{NoSOI}(\tau)^T \bullet \frac{\vec{F}^{ideal}(\tau)}{M_{S/C}(\tau)} \div 10^6 d\tau \end{aligned} \quad (2)$$

The measure of energy was the answer to address SOI critical burn restart issues that plagued a previous pre-launch smart burn algorithm implementation. The variables for current spacecraft velocity (V_{NoSOI}) relative to Saturn (without the burn), which utilized existing cubic spline conic vector propagation, and spacecraft acceleration (A_{SOI}) as measured by the accelerometer (due to the burn), already existed to calculate the estimated change in specific energy using Eq. (1). For burn cutoff, ideal energy achieved was used as the terminator given min and max target values. The software hooks for Main Engine force magnitude (F^{ideal}) and on-board time-varying best estimate of spacecraft mass ($M_{S/C}$) already existed for ideal changes in specific energy using Eq. (2). In addition to the algorithm verification, this was the first time a time-varying burn vector was used. Other first-time events were using a new command to issue the burn and the downlink of new telemetry that capture burn details. The success of TCM-19b provided the confidence that SOI was technologically achievable using an energy burn cutoff algorithm.

The concept of post-update checkout sequences should always be considered after an upload of FSW. Cassini operations performs tedious ground testing before uplink of FSW or sequences of extreme importance. Steps are taken on the ground to mimic the actual spacecraft environment. However, nothing beats the real thing. The Cassini program tries to eliminate as many first time events as possible on new software. This provides the time to adequately monitor and evaluate FSW functionality before being invoked to support spacecraft and science sequences. There were two major checkouts after the full A7.7.6 and A8.6.5 in-flight uploads.

For A7.7.6, the checkout activities lasted twenty days. Checkout activities verified that all phasing was correct. RWA performance in both rate and attitude control modes were per expectations, and per-axis attitude control errors were better than the requirements. With the validation of FSW capabilities, remember to address the closure of applicable flight rules. Sixteen flight rules were deleted and eleven others were modified.

For A8.6.5, the checkout activities spanned over a month. New FSW algorithm functionality performance was verified to support upcoming critical sequence activities such as SOI and critical ring plane crossings. Waivers were written to perform checkout activities on backup ACS hardware.

In-flight checkouts were used to verify and validate new FSW functionality performance, and interactions with backup devices which could be potentially invoked in fault conditions for critical events. The in-flight data collected from these activities were invaluable and provided confidence by experience that FSW could perform the necessary tasks when required.

F. Software Data Definitions and Memory Protection

The difference between *variables* and *constants* is that constants are validated by a checksum routine and variables are not. For ACS FSW, the checksum routine is only exercised once during the FSW executive startup's RVB routine. Execution placement of checksum routines may hide issues which will not become evident until exercised during some future event. This allows the possibility for a constant to be modified accidentally without showing any ill-effects in the software. The only time this would be caught in FSW is after an AFC reset where the RVB routine is executed. When a checksum fails, the AFC keeps the FSW in ROM and does not transition to RAM. If an actual anomaly occurred, this could complicate recovery or exercise unnecessary tiers of fault protection to load FSW from the SSR. Newer missions may reset their computers every time an update is performed, which will catch checksum errors early. For missions not practicing this technique, management of checksum data needs to be monitored.

Telemetry parameters such as scale factors and range definitions are defined as constants. Pre-launch perception is that these parameters need to correctly account for their integrity by the method of checksums. In post-launch scenarios telemetry resolutions or ranges may adjust to changing mission objectives. It may be more practical to implement telemetry parameters as variables which are not checksummed. Telemetry is monitored daily and has high visibility. If a problem were to occur, it could be tracked and easily changed without the multitude of activities associated with updating constants. Major efforts are focused on the verification of parameter selections; however, extra attention is needed when identifying FSW data as variables or constants.

Legal range checking is another method to insure that erroneous data values are not used in commands or FSW conditional checks. There are instances where the FSW does legal range checking on some commands but not others. Attention to consistency needs to be practiced. All of the ground parameter commands do not implement range checking. While the original intent may have wanted to give flexibility to updating commandable parameters, if care is not practiced, an ill-chosen value could result in undesired FSW interactions. Pros and cons need to be weighed between flexibility and parameter range checking to ensure that valid values are within parameter operating ranges. Solutions must be well documented in command dictionaries. Whether the FSW does this range checking or not, the ground support software must not assume that FSW will catch commanding errors. Ground software must be resilient enough to range check all commands sent to the spacecraft.

G. Recovery Data

Many patch table updates were performed to insure that the default RCS thruster magnitudes were updated to closely match actual physical (active) values. The default updates were performed to reduce a mismatch between FSW and physics, and insures that certain thruster-related FP monitors not hinder recovery efforts. RCS thrusters are the key attitude control mechanism during recovery efforts i.e., performing Sun search. ACS FSW has a recovery data set that is stored in CDS. If the prime AFC is reset or a swap of AFCs occurs, then the SUROM will request the recovery data from CDS when ACS FSW is reinitialized. Recovery data consists of key data that takes a snapshot of the most recent state. If the data requested in the recovery set are corrupt, default values will be used. There are two sets of defaults, one for FSW contained in the SSRs and another for FSW contained in the BAIL.

If RCS thruster magnitudes and Safing attitudes (and other data related to mission phases or data key to spacecraft recovery) were captured in recovery data, a robust management of recovery data could ease or eliminate the need to update default values via the patch table. Having methods to capture previous states of FSW parameters and making them available at FSW initialization are invaluable, especially during recovery efforts. Updates to key parameters via recovery data could replace FSW parameter patches if a method to modify or add additional recovery data were in place. This would simplify the operations process by eliminating the need to generate new FSW builds to update default parameters.

H. Software State Monitoring

There are several instances in the FSW, where the masking of FP error monitors disables and bypasses the entire monitor code block. While the original intent may have wanted monitors to be completely disabled, in-flight operations experience has revealed that it may be better to allow error monitor code block execution (which provides greater visibility into FSW and error monitor thresholds) without reporting an opinion or activation of a response script. This effect would still disable the monitor. The exit criteria for masking should not completely bypass the error monitor code or code that will give insight into threshold encroachment. For Cassini, an example of this would be high-water mark data. There are several error monitors that if masked, simply bypass code which captures high-water mark data which would be very useful to reconstruct events of interest during the masked event.

Beware of mismatches which may occur in FSW when states change unexpectedly. Ideally, this should not occur in an object oriented paradigm. However, this is an example where minimal coupling of objects and data scope enforcement were loosely coupled. FSW Fault Protection responses related to hardware anomalies can load-shed (power off) non-vital hardware such as RWAs that are non-vital in recovery efforts.¹³ Software associated with the hardware states may no longer update data which other objects use or monitor which may cause stale state data. The instantaneous powering off of hardware devices without proper momentum unloading can result in a discrepancy between physical hardware properties and the FSW knowledge. Therefore, software variables may contain the last known values before hardware is powered off unexpectedly. An AFC reset will initialize software variables to and initial state which also causes a discrepancy for non-zero physical

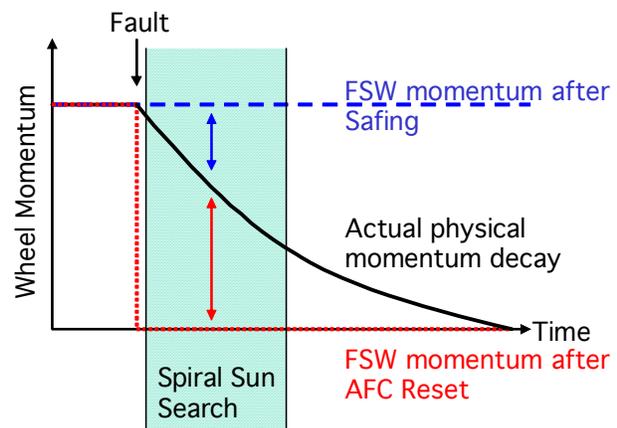


Figure 19. Graphical Concept of RWA FSW Phantom Momentum. Illustrates two scenarios of FSW discrepancy. Blue arrow discrepancy occurs when a Safing or FP-related RWA to RCS transition is invoked, Red arrow discrepancy occurs when an AFC reset is invoked.

hardware states. Figure 19 illustrates the discrepancies. Software logic should be more robust in checking states before utilization of interdependent data. There are operational work-arounds to address these issues, which do not require a fix to FSW logic to eliminate the “phantom” or erroneous data sources.

The more autonomous and complex the software becomes, the harder it becomes to verify and validate. While state machines and object oriented methodologies simplify architecture and exhibit attributes of extensibility, modifiability, and maintainability – once implementation diverges from this paradigm, interfaces become nebulous and testing becomes more difficult. The practice of implementing simplicity in a complex environment is an attribute which a developer needs to embrace and cultivate.

XI. Conclusion

Cassini made history becoming the first spacecraft to orbit Saturn. For the general public, this was the first time Cassini became a household word. For the JPL community, Cassini was the center of technical achievement for over ten years of design and development. It brought together the best of the best associated with hardware, software, and support personnel. During the operations phase, the philosophy and commitment to excellence made Cassini one of the most successful missions ever flown. With Cassini nearing the end of its mission, this paper reflects and celebrates the numerous ACS FSW design, development, test, and operational achievements.

Initial stages of FSW prototyping began in 1990. The first official delivery of FSW was 30 September 1994 with version A1.1.0. The accelerometer and Sun sensor hardware managers were the only modules not stubbed. Over 100 builds later, the last scheduled FSW build – version A8.7.7, will be uplinked to the spacecraft in June 2009 to support the remainder of the extended mission. The final build will contain a twenty-year heritage of AACS control architecture and state machines objectified into commanders, controllers, estimators, software managers, hardware managers, fault managers, and glue code. Testament to the FSW success is the abundance of test results review findings, procedures, processes, and performance. The sustainment of methodologies and rigorous adaptations to improve methodologies in the operations phase have resulted in lessons learned that give insight into design and development philosophies, verification of first time events, memory protection techniques, and future software design strategies, all which may not be obvious without in-flight experience.

While this paper contains a deluge of information about ACS FSW from its beginning to its operability in flight, hopefully it will enlighten – without too much bewilderment – the amount of work and effort needed to make Cassini a success.

Acknowledgments

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Reference to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

Jay Brown acknowledges everyone who worked on the Cassini AACS team, and the collaboration of analysts and other subsystems that made and continue to make Cassini the most successful interplanetary mission to date. There are just too many people who have supported Cassini ACS FSW over the years to list individual names. Instead, the author will focus acknowledgments to the pre and post FSW development teams: Linda Bagby, Harry Balian, Mary Bruskotter, John Buchman, Larry Chang, Steven Cheung, Ken Clark, Martin Gilbert, Kim Gostelow, Chris Granger, Danny Lam, Mary Lam, Scott Peer, George Shesby, Marek Tuszynski, John Vande Wege, Eric Wang, Garth Watney, and Katia Zarnegar. Acknowledgments to key FSW architects: Dr. Doug Bernard, G. Mark Brown, John Hackney, Dr. Bob Rasmussen, and Dr. Edward Wong. Special thanks to Dr. Allan Y. Lee, Glenn Macala, Juan Hernandez, Karen Lum, Peter Meakin, Tom Burk, Dave Beach, Cindy Huynh, and Dr. Gurkirpal Singh for providing their knowledge and support to the writing of this paper. Jennifer Schuray illustrated the Cassini spacecraft represented in Figure 1. Enrico Attanasio, Kim Gostelow, Mary Lam, Dr. Bob Rasmussen, Dr. Doug Bernard, and John Hackney created the final design of Figure 12 – the FSW Mode Transition Diagram.

References

¹Webster, J.L., “The Cassini Spacecraft Design and Operations,” *Space Technology & Application International Forum*, Albuquerque, NM, Feb. 13-16, 2005.

²Jaffe, L.D., and Herrell, L.M., “Cassini/Huygens Science Instruments, Spacecraft, and Mission,” *Journal of Spacecraft and Rockets*, Vol. 34, No. 4, July-August 1997, pp. 509-521.

- ³Lee, A.Y., and Hanover, G., "Cassini Attitude Control System Flight Performance," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2005-6269, San Francisco, CA, Aug. 15-18, 2005.
- ⁴Draper, R.F., "The Mariner Mark II Program," *AIAA 26th Aerospace Sciences Meeting*, AIAA-88-0067, Reno, NV, Jan. 11-14, 1988.
- ⁵Hackney, J.C., Bernard, D.E., and Rasmussen, R.D., "The Cassini Spacecraft: Object Oriented Flight Control Software," *Advances in the Astronautical Sciences*, Vol. 81, 1993, pp. 211-236.
- ⁶Wong, E.C., and Breckenridge, W.G., "An Attitude Control Design for the Cassini Spacecraft," *AIAA Guidance, Navigation and Control Conference*, AIAA-1995-3274, Baltimore, MD, Aug. 7-10, 1995, Technical Papers. Pt. 2 (A95-39609 10-63), Washington, DC, American Institute of Aeronautics and Astronautics, 1995, pp. 931-945.
- ⁷Hackney, J.C., et al., "Cassini Project Attitude and Articulation Control Subsystem Software Specification Document," JPL D-13264, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Jun. 1993 – Nov. 2002 (unpublished).
- ⁸Harel, D., "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* 8, June 1987, pp. 231-274.
- ⁹Brown, G.M., Hackney, J.C., Rasmussen, R.D., and Zarnegar, K., "Storing and Loading the Flight Software for Cassini's Attitude and Articulation Control Subsystem: A Fault Tolerant Approach," *15th AIAA/IEEE Digital Avionics Systems Conference*, Atlanta, GA, Oct. 27-31, 1996.
- ¹⁰Wang, E.K., and Brown, J.M., "Cassini Test Methodology for Flight Software Verification during Operations," *15th AIAA Guidance, Navigation and Control Conference and Exhibit*, Hilton Head, SC, Aug. 20-23, 2007.
- ¹¹Hackney, J.C., et al., "Cassini Project Attitude and Articulation Control Subsystem Software Test Plan," JPL D-13265, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Jun. 1993 – Oct. 2002 (unpublished).
- ¹²Brown, G.M., Bernard, D.E., and Rasmussen, R.D., "Attitude and Articulation Control for the Cassini Spacecraft: A Fault Tolerance Overview," *14th AIAA/IEEE Digital Avionics Systems Conference*, Cambridge, MA, Nov. 5-9, 1995.
- ¹³Brown, G.M., and Johnson, S.A., "An Overview of the Fault Protection Design for the Attitude Control Subsystem of the Cassini Spacecraft," *Proceedings of the 1998 American Control Conference*, Vol. 2, Jun 24-26, 1998, pp. 884-898.
- ¹⁴Badaruddin, K.S., Hernandez, J.C., and Brown, J.M., "The Importance of Hardware-In-The-Loop Testing to the Cassini Mission to Saturn," *IEEE Aerospace Conference*, Track 12.0503, Paper 1231, Big Sky, MT, Mar. 3-10, 2007.
- ¹⁵Brown, J.M., Lam, D.C., Chang, L., Burk, T.A., and Wette, M.R., "The Role of the Flight Software Development System Simulator throughout the Cassini Mission," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2005-6389, San Francisco, CA, Aug. 15-18, 2005.
- ¹⁶Lee, A.Y., and Brown, J.M., "A Model-Based Thruster Leakage Monitor for the Cassini Spacecraft," *Proceedings of the American Control Conference*, Philadelphia, PA, June 1998, pp. 902-904.
- ¹⁷Lam, D.C., Friberg, K.H., Brown, J.M., Sarani, S., and Lee, A.Y., "An Energy Burn Algorithm for the Cassini Saturn Orbit Insertion," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2005-5994, San Francisco, CA, Aug. 15-18, 2005.
- ¹⁸Cervantes, D., Badaruddin, K.S., and Huh, S.M., "Integrated Testing of the Cassini Saturn Orbit Insertion Critical Sequence," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2005-6272, San Francisco, CA, Aug. 15-18, 2005.
- ¹⁹Allestad, D.L., Standley, S.P., Chang, L., and Bone, B.D., "Systems Overview of the Cassini-Huygens Probe Relay Critical Sequence," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA-2005-6388, San Francisco, CA, Aug. 15-18, 2005.
- ²⁰Feldman, A.W., Brown, J.M., Wang, E.K., Peer, S.G., and Lee, A.Y., "Reconstruction of Titan Atmosphere Density using Cassini Attitude Control Flight Data," *17th AAS/AIAA Space Flight Mechanics Meeting*, AAS-07-187, Sedona, AZ, Jan. 28-Feb. 01, 2007.
- ²¹Chang, L., Brown, J.M., Barltrop, K.J., and Lee, A.Y., "Use of Guidance and Control Test Cases to Verify Spacecraft Attitude Control System Design," *AAS/AIAA Space Flight Mechanics Meeting*, AAS 02-122, AIAA, Washington, DC, Jan. 2002.
- ²²Hihn, J.M., "Managing with Software Metrics," *JPL Internal Class given by the JPL Education and Training Consortium*, Pasadena, CA, Jul. 23-24, 2008.