The Legacy of Space Shuttle Flight Software

Christopher J. Hickey[1] and James B. Loveall[2]
NASA Johnson Space Center, Houston, Texas, 77058

James K. Orr[3] and Andrew L. Klausman[4]
United Space Alliance, Houston, Texas, 77058

The initial goals of the Space Shuttle Program required that the avionics and software systems blaze new trails in advancing avionics system technology. Many of the requirements placed on avionics and software were accomplished for the first time on this program. Examples include comprehensive digital fly-by-wire technology, use of a digital databus for flight critical functions, fail operational/fail safe requirements, complex automated redundancy management, and the use of a high-order software language for flight software development. In order to meet the operational and safety goals of the program, the Space Shuttle software had to be extremely high quality, reliable, robust, reconfigurable and maintainable. To achieve this, the software development team evolved a software process focused on continuous process improvement and defect elimination that consistently produced highly predictable and top quality results, providing software managers the confidence needed to sign each Certificate of Flight Readiness (COFR). This process, which has been appraised at Capability Maturity Model (CMM)/Capability Maturity Model Integration (CMMI) Level 5, has resulted in one of the lowest software defect rates in the industry. This paper will present an overview of the evolution of the Primary Avionics Software System (PASS) project and processes over thirty years, an argument for strong statistical control of software processes with examples, an overview of the success story for identifying and driving out errors before flight, a case study of the few significant software issues and how they were either identified before flight or slipped through the process onto a flight vehicle, and identification of the valuable lessons learned over the life of the project.

I. Introduction

The Space Shuttle Flight Software has a unique role in aerospace history. With the STS-135 landing, the Space Shuttle Primary Avionics Software System (PASS) has completed over thirty years of operational service as the "brains" of the fly-by-wire Space Shuttle Vehicle. All told, PASS has been in control for over 1300 days of orbital flight or over 20,000 orbits. This means that some of the highest execution rate software routines in PASS have executed over 2 billion times in life critical situations. And, in an industry that has experienced many prominent and critical software failures, PASS has successfully completed its mission with a perfect record of never contributing to a loss of mission, crew, or vehicle. This was a remarkable accomplishment for a system that developed and deployed innovative new technologies in order to meet stringent Space Shuttle system requirements. The Space Shuttle Avionics and PASS legacy includes:[5]

- First successful attempt to incorporate a comprehensive fail operational/fail safe concept in an avionics system.

[1] Software Engineer, Space Shuttle Flight Software Office, Johnson Space Center, 2101 NASA Parkway, Mail Stop MS4.
[2] Division Chief Engineer (DCE) for shuttle software, Avionic Systems Division, Johnson Space Center, 2101 NASA Parkway, Mail Stop EV7.
[3] Flight Software Chief Engineer, United Space Alliance, 600 Gemini, Houston, TX 77058 (search James Orr Space Shuttle, see LinkedIn for current contact information)
[4] USA MEDS Project Manager, Flight Software Element Coordination, United Space Alliance, 600 Gemini, Houston, TX 77058
[5] Hanaway, J. F., and Moorehead, R. W., "Space Shuttle Avionics System," NASA SP-504, 1989.

- Pioneered the development of complex redundancy management techniques which rival modern expert systems.
- First aerospace system to use digital data bus technology to perform flight critical functions.
- First operational system to utilize a high order software language for onboard software.
- First operational aerospace program to make extensive use of flight software program overlays from mass memory storage devices to expand the effective size of main computer memory.
- First system to integrate the flight control function with the rest of the avionics functions,
- First use of digital fly-by-wire technology in an operational atmospheric flight application.
- First use of a distributed system architecture in spaceflight to provide a separate Systems Management function.
- First avionics system to use a multifunction crew display and interface approach.
- First avionics system to provide extensive operational services to onboard non-avionics systems.
- First use of synchronization of multiple computers running identical software for fault detection
- First software development organization to be assessed as Capability Maturity Model (CMM) Level 5

When a disciplined software maintenance approach is enforced, critical software gets better as it ages. In the case of PASS, an extremely mature repeatable software engineering approach was evolved that has been recognized as one of the pre-eminent in the field of critical software and was lauded as the first organization in the world to achieve CMM Level 5. Throughout the lifetime of PASS, the technical, process, and organizational causes of each uncovered error have been systematically identified and corrected. After thirty years of refinement, the result is an exceptionally reliable software system where the only remaining errors are in extremely remote paths that are scenario/failure intensive. Considering the over 2 billion operational executions that some PASS software modules have completed, a one in a million pathway would have executed over 2000 times in an operational environment. Although it can be hypothesized that new technologies and methodologies will lead to higher initial quality in critical software systems, it seems clear that extended operational shelf-life is a necessary ingredient before a complex software system can achieve the ultra-reliable status that PASS enjoys at its retirement.

II. PASS Overview

The Space Shuttle was the first manned space vehicle that was totally reliant on the embedded computer system. As a digital fly-by-wire system, virtually no critical activity could be accomplished without the successful operation of the hardware and software of the On-Board Data Processing System (DPS). A short DPS outage during critical operations (Ascent, Entry, and Orbital Proximity Operations) would almost certainly lead to Loss of Crew and Vehicle (LOCV). For a nominal Space Shuttle ascent, no crew inputs are required. For a nominal Space Shuttle entry, only manual landing gear and air data probe deploy are required although the crew has manually flown the final approach and landing phase for every mission.

With the Space Shuttle DPS architecture, all sensor inputs are read into the DPS system and (almost) all effector commands are output from DPS system. If the DPS system fails to output a correct command on-time or outputs an erroneous command, the results could be catastrophic. So, the reliability requirement for the DPS system is essentially flawless performance for all approved scenarios and configurations.

III. PASS Design Goals and Philosophy[6]

The initial design goals for the overall shuttle avionics system in turn drove the design of the PASS. The significant mission requirements and vehicle constraints that drove this overall avionics design included the following:

- Reliability through two fault tolerant fail operational/fail safe capabilities
- Redundancy through independent parallel strings of avionics hardware
- Challenging control authority requirements including unpowered runway landing
- Multiple abort capabilities
- Complex on-orbit operations for a wide variety of payloads and missions

[6] Hanaway, J. F., and Moorehead, R. W., "Space Shuttle Avionics System," NASA SP-504, 1989.

- Autonomy from the ground for comprehensive onboard subsystem management
- Use of digital data busses to reduce wire volume and weight

One of the requirements with the more significant impact on avionics and software design was the fail operational/fail safe requirement. This meant that the avionics system must remain fully capable of performing the operational mission after any single failure and fully capable of returning safely to a runway landing after any two failures in the same subsystem. This drove the use of multiple avionics strings each independent from a reliability standpoint but each with equivalent capability. The Fault Detection, Isolation and Reconfiguration (FDIR) requirements called for comparing actual operational data from one subsystem or device with similar data produced by peer subsystems/devices operating in parallel to perform the same function. A minimum of three strings was required to identify a diverging or disabled unit, and a fourth string was required to accommodate a second failure in the same area. This led to the design for quadruple redundancy and four independent strings.

Another significant requirement which affected avionics design was support for autonomous operations to reduce dependency on the ground. Much of the subsystem telemetry data, which had been sent only to the ground in previous programs, was to be processed and provided to the crew onboard. Extensive automatic subsystem monitoring and control was also required to reduce dependence on ground support.

The Space Shuttle vehicle was an unstable airframe which could not be flown manually even for short periods of time during either ascent or entry aerodynamic flight phases without full-time flight control stability augmentation. There were also precise vehicle sequencing requirements for such events as Space Shuttle Main Engine (SSME) and Solid Rocket Booster (SRB) ignition, launch pad release and lift-off operations, and SRB and External Tank (ET) stage separation which must occur within milliseconds of the correct time. These requirements drove to a completely digital fly-by-wire automated system. Because sensors and effectors were to be distributed all over the vehicle, the weight of the wire to carry all of the necessary signals became a significant factor. Therefore, the use of multiplexed digital data busses was baselined.

In order to support many missions throughout the years, and to keep software maintenance costs down, the software system was required to be data-driven and reconfigurable from flight-to-flight, mission-to-mission, payload-to-payload without requiring software logic changes, re-compilation and extensive re-test.

Obviously, the flight software team adopted the goals of high quality, error free, reliable, robust, efficient, and maintainable software. This section provides an overview of the requirements and design environment to which the original avionics and software team was exposed. Further sections of this paper will expand upon these design concepts.

IV. PASS Architecture

The Primary Avionics Software System (PASS) was a complex system designed to drive the Data Processing System (DPS) of the Space Shuttle. The DPS was centered around 5 AP-101 General Purpose Computers (GPCs) that are linked together via Inter-Computer Communication (ICC) busses. Initially, the GPCs were IBM model AP-101B computers that consisted of separate hardware boxes for the Central Processing Unit (CPU) and Input/Output Processor (IOP) which between them contained 416 Kilobytes (KB) of ferrous core (persistent) memory. In 1989 (the first flight was in 1991), the GPCs were upgraded to the AP-101S version which featured a combined CPU/IOP hardware box and 1024 KB of CMOS memory. Typically, GPC1 – GPC 4 were dedicated to running PASS software and GPC 5 was reserved for running the Back-up Flight System (BFS). BFS was an independently developed software system capable of safely completing Ascent or Entry in the event of a PASS software failure. BFS normally performs some Systems Management and Monitoring functions during Ascent and Entry. Both PASS and BFS were custom-built and coded in assembler language and HAL/S. HAL/S is a NASA-developed high-level language that compiles into very efficient assembly language code and is a real-time version of IBM's Programming Language/1 (PL/1). For example, the PASS Flight Computer Operating System (FCOS) was coded in assembler language and the applications are coded in HAL/S (e.g., Guidance, Navigation, and Control or GNC).

PASS was comprised of the System Software (SSW) that provides system services for all PASS configurations and a series of flight phase-specific application loads that could be run on any GPC. Any time PASS was running in multiple GPCs, the GPCs formed a Common Set (CS) where state data is exchanged between PASS GPCs at 6.25

times a second and enables the identification of certain hardware or software failures. Multiple copies of the Ascent, Orbit, and Entry GNC loads could run simultaneously to form a Redundant Set (RS), where software execution was synced at rates of ~400 times a second, to enable rapid detection of a larger set of software errors and allow the erroneous GPC to be voted out quickly by the remaining GPCs. PASS was typically run in a 4 GPC redundant set through all of Ascent and Entry and in a two or three GPC redundant set during critical Orbit operations. During most orbital periods, the PASS GNC software was run in one GPC and the PASS Systems Management (SM) software was run in one GPC. The remaining three GPCs (including the BFS GPC) were powered down for efficiency.

Designation	**Description**	**Purpose**
OPS 0	GPC memory utilities	Basic GPC system utility software
GNC OPS 1	Ascent GNC	Controls all GNC systems from shortly before launch through Orbit
GNC OPS 2	Orbit GNC	Controls all GNC systems for all Orbital Activities
GNC OPS 3	Entry GNC	Controls all GNC systems from shortly before De-Orbit through Landing
GNC OPS 6	RTLS GNC	Controls all GNC Systems when a Return to Launch Site (RTLS) abort is declare – this OPS is co-resident with OPS 1
PL 9	VU Mass Memory utility	Provides the ability to load or alter Mass Memory
SM OPS 2/4	Systems Management	Includes software required to monitor and manage all non-GNC systems (e.g., electrical power, payloads) during Orbital phase
VU OPS 8	On-Orbit Checkout	Includes Orbit GNC and a Vehicle Utility (VU) systems test for all De-Orbit, Entry, and Landing systems
VU OPS 9	Pre-Launch and Post-Landing Checkout	Includes VU software required to prepare orbiter for launch and reconfigure orbiter after landing (e.g., IMU Calibration)

Figure IV-1 PASS Software Loads (Mass Memory Overlays)

In addition to the 5 ICC busses that connect the GPCs to each other, each GPC was connected to 19 other busses which provide connectivity over virtually the entire Space Shuttle vehicle. Eight of these busses were wired to Flight Critical Multiplexer/De-Multiplexers (MDMs) Flight Forward 1-4 (FF1-FF4) and Flight Aft 1-4 (FA 1-4). These 8 MDMs connected the GPCs with all of the redundant GNC sensors and effectors, and are channelized to maintain redundancy if an MDM should fail. Other busses connected the GPCs with the launch pad, Solid Rocket Boosters, payloads, orbiter health and environmental hardware, mass memories, and user interfaces (e.g., on-board displays/keyboards, telemetry).

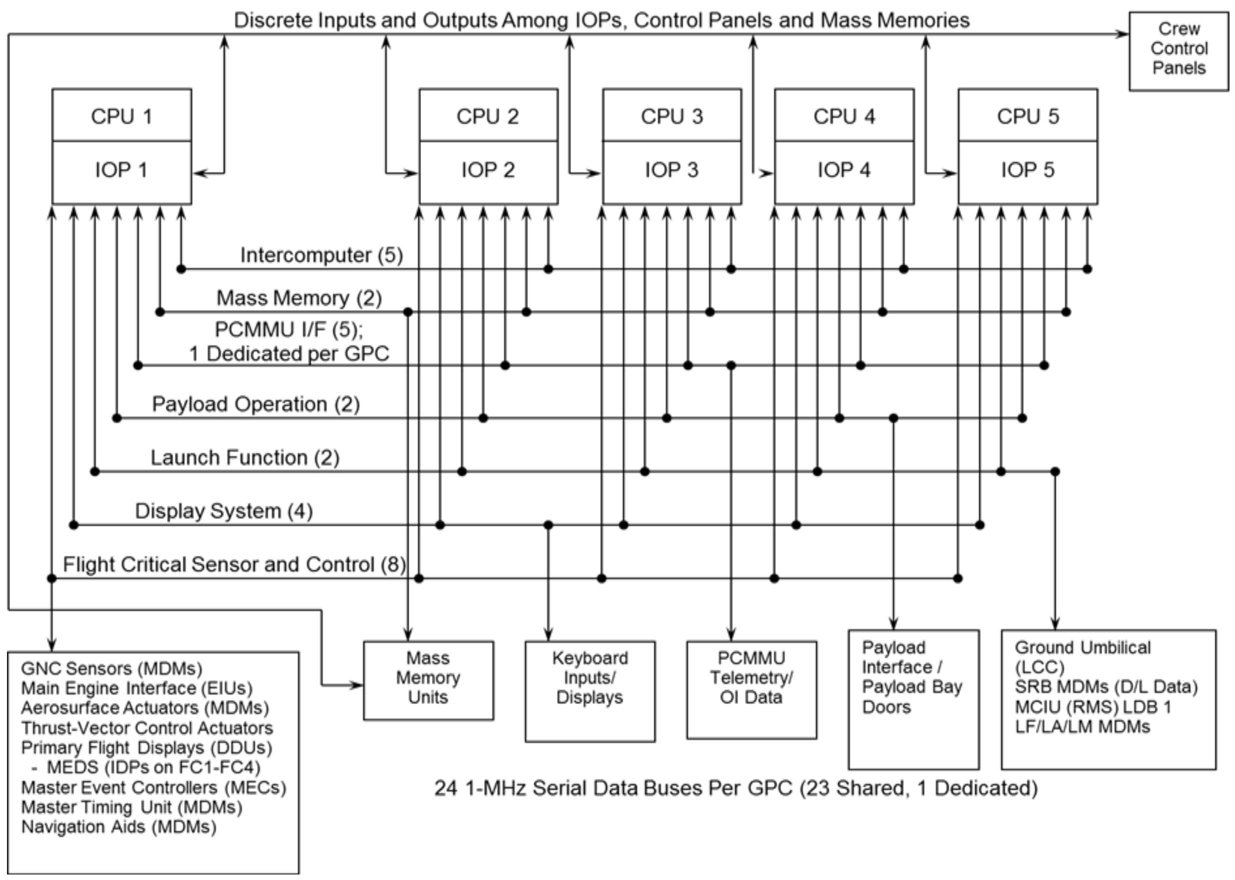

Discrete Inputs and Outputs Among IOPs, Control Panels and Mass Memories
Crew Control Panels
CPU 1
IOP 1
CPU 2
IOP 2
CPU 3
IOP 3
CPU 4
IOP 4
CPU 5
IOP 5
Intercomputer (5)
Mass Memory (2)
PCMMU I/F (5); 1 Dedicated per GPC
Payload Operation (2)
Launch Function (2)
Display System (4)
Flight Critical Sensor and Control (8)
GNC Sensors (MDMs)
Main Engine Interface (EIUs)
Aerosurface Actuators (MDMs)
Thrust-Vector Control Actuators
Primary Flight Displays (DDUs)
- MEDS (IDPs on FC1-FC4)
Master Event Controllers (MECs)
Master Timing Unit (MDMs)
Navigation Aids (MDMs)
Mass Memory Units
Keyboard Inputs/ Displays
PCMMU Telemetry/ OI Data
Payload Interface / Payload Bay Doors
Ground Umbilical (LCC)
SRB MDMs (D/L Data)
MCIU (RMS) LDB 1
LF/LA/LM MDMs
24 1-MHz Serial Data Buses Per GPC (23 Shared, 1 Dedicated)

Figure IV-2 GPC Digital Databus Overview

The PASS System Software (SSW) was a custom, interrupt driven operating system that consists of three components:

- Flight Computer Operating System (FCOS) provided input/output, GPC interface, and redundant/common set management services.
- System Control (SC) provided PASS initialization, memory and bus reconfiguration, and ICC bus services.
- User Interface (UI) provided keyboard/display interface, uplink/downlist, and caution and warning services.

The PASS applications consisted of three components that were comprised of over 450 distinct applications:

- Guidance, Navigation, Control (GNC) provided the software that issues all of the commands required to insure that the Orbiter safely gets from the launch pad to the required orbit, from the orbit to the rendezvous targets, and from orbit to the desired runway.
- Systems Management (SM) provided subsystem monitoring and control, Remote Manipulator System (RMS) interface, payload interface, and antenna management.
- Vehicle Utility (VU) provided the ground interface to the orbiter on the launch pad, Pre-Launch support functions, Mass Memory interface, on-orbit sub-system checkout, and post-landing support.

PASS was designed to be efficiently reconfigured from flight to flight. A large set of flight-specific data (Initial Loads or I-loads) was validated and installed into the flight software (FSW) via automated processes to create a flight load. This was done by replacing the stored default value for a particular parameter with the flight specific I-Load value to create the flight load. Before it flew, this flight load underwent flight-specific testing and was used in post-release simulations. Approximately ten days before flight a small set of low-risk updates were allowed (after re-testing). And, finally, an automated process was executed on the day of launch to update a small set of day-of-launch parameters (e.g., winds) to customize the load for the conditions on that particular day.

Until 1997, the cockpit had a crew display capability that included 4 “Green Screen” display and processing units and a set of mechanical dedicated displays. Then a “glass cockpit” system was installed, called the Multifunction Electronics Display Subsystem (MEDS), which featured 11 color flat panel screens and a large increase in processing power. MEDS replaced all of the mechanical dedicated displays (e.g., Altimeter, Attitude Directional Indicator) with graphical/color versions that could be configured on the 11 screens based upon crew preferences. In 2007, the MEDS-based Ascent and Entry Bearing displays were released as the last major upgrade to PASS. These displays provided improved crew situational awareness by graphically displaying the orbiter position, progress, and projected trajectory overlaid on a series of maps in order to assist with abort determination.

When multiple PASS GPCs were running, SSW insured that only one GPC could command a bus at a given time. On Ascent and Entry, each of the 4 GNC GPCs were assigned on flight critical string (FF/FA combination). Although the system provided the flexibility for each GPC to command any and all busses depending upon the application load, the typical assignments for Ascent and Entry (when all four GPCs were running GNC loads) were:

- GPC 1 is assigned FF1 and FA1
- GPC 2 is assigned FF2 and FA2
- GPC 3 is assigned FF3 and FA3
- GPC 4 is assigned FF4 and FA4

If a GPC were to fail, insight to all measurements on the busses the GPC was commanding and the ability to send any commands channelized to those busses was lost unless action was taken to transfer control of those busses to a healthy GPC. However, the PASS Redundancy Management (RM) software was designed to seamlessly adjust to loss of measurements without impacting the mission.

The Space Shuttle’s reliability requirements were based upon a Fail Operational/Fail Safe (FO/FS) system requirement. This requirement said that the vehicle could tolerate any single failure to any subsystem without impacting its ability to complete the mission and any two failures to the same subsystem without impairing the ability to safely return crew and vehicle to earth. The FSW’s Fault Detection, Identification, and Reconfiguration (FDIR) logic was a trailblazer in the field of software-controlled RM. In order to achieve these requirements, it was essential that the FSW be capable of quickly and accurately identifying and responding to a wide variety of system and sub-system failures including those to itself or its supporting hardware. While recognizing a failure when three independent measurements were available was relatively straightforward (if one measurement behaves differently from the other two, it is bad), it was much more challenging to identify the failure when two measurements were diverging. This often required incorporation of expectations and secondary indications to infer the range of expected values.

The other RM challenge was identifying a software or DPS hardware failure. How do you recognize when the system that is detecting failures is failing? It was this case that drove much of the system architecture. The DPS system was designed with the capability of up to four independent General Purpose Computers (GPCs) independently running four copies of the same software. And, based upon configuration, each of these GPCs was given the ability to issue a subset of the commands. These commands can be as simple as each GPC controlling a separate piece of hardware (e.g., RCS jets) or the GPC inputs being summed by the hydraulics to determine a rate of movement (e.g., elevons).

But RM’s most significant technical leap came when these four GPCs were synchronized together in a Redundant Set (RS) in order to identify software or DPS hardware failures. Approximately 400 times a second, FCOS entered a sync routine where all GPCs must stop and compare states via 3 discrete sync lines cross-strapped to each GPC. This was a functional gate where all GPCs must stop and wait for all members of the RS to arrive. If one or more did not arrive in a reasonable amount of time, they were voted out of the set. If all GPCs arrived, but one did not have the same state data as the others, it was voted out of the set. Once these comparisons were completed, the gate was lifted and the GPCs left the sync point together and continued until the next sync point was reached. A failed GPC was halted, via manual switch, to prevent issuing of erroneous commands. But, unless the DPS system was reconfigured to pick up the busses lost by the failed GPC, the capability to communicate with the hardware the failed GPC was commanding was lost.

Redundancy Management Example

The DPS can be configured so that each of the GPCs is commanding one of the four aft right firing Reaction Control Jets. A basic redundancy principle is that identical inputs going though identical processing logic will yield identical outputs. As the four GPCs independently execute the Guidance, Navigation, and Control algorithms from the same set of inputs, they should all arrive at the same control decision. In this case, if they decide that a small +Yaw correction is required, they will all command one aft right firing jet to fire (the highest priority jet – R2R). While all four GPCs are issuing a command to fire R2R, the Flight Computer Operating System (FCOS) software is actually only responding to the GPC 2 command since it controls the data bus that R2R is attached to. If a large (4 jet) correction is required, all four GPCs will command all four of the +Yaw jets to fire. But, in actuality, GPC1 is only controlling R1R, GPC2 is controlling R2R, GPC3 is controlling R3R, and GPC4 is controlling R4R.

Back-up Flight System Overview

The Back-up Flight System (BFS) was developed in parallel to PASS originally by Rockwell International (later part of the Boeing Company) as a mitigation for generic PASS failure (defined as a single software error that could cause all four PASS GPCs to stop functioning correctly). BFS was initially intended to support only the Approach and Landing Test (ALT), but was extended through STS-4, and, finally, extended indefinitely. Some features of BFS included:

- Ran in a single GPC as a single load that supports Automated Ascent and Manual Entry (no orbit support).
- Able to "track" PASS execution by monitoring bus activity in order to be prepared for engagement at any moment.
- Provided System Management functionality for Ascent and Entry (even when not engaged).
- Designed as Zero Fault tolerant (some exceptions made).
- Did not support Contingency Aborts (2EO / 3EO).

V. PASS Organizational Structure

The PASS organization existed from when NASA awarded IBM Corporation the first independent Shuttle software contract on March 10, 1973 through the landing of the last Space Shuttle mission on July 21, 2011. At the end of the Space Shuttle Program, PASS was part of United Space Alliance.

During all of this time, the organization had the same key functions. These were (1) Systems Engineering / Requirement Analysts, (2) System Software Development, (3) Application Software Development, (4) Functional Verification, (5) Performance Verification, (6) Project Office / Project Coordination, and (7) Test and Operations / Customer Support. In addition to the direct PASS organization, the overall organization always included a separate organization that developed the Software Development Laboratory where development, reconfiguration, and verification was performed.

Figure V-1 shows how the PASS organization performed as an integrated product team with one role having primary responsibility during major software development phases while being supported by other key roles including participation in product inspections.

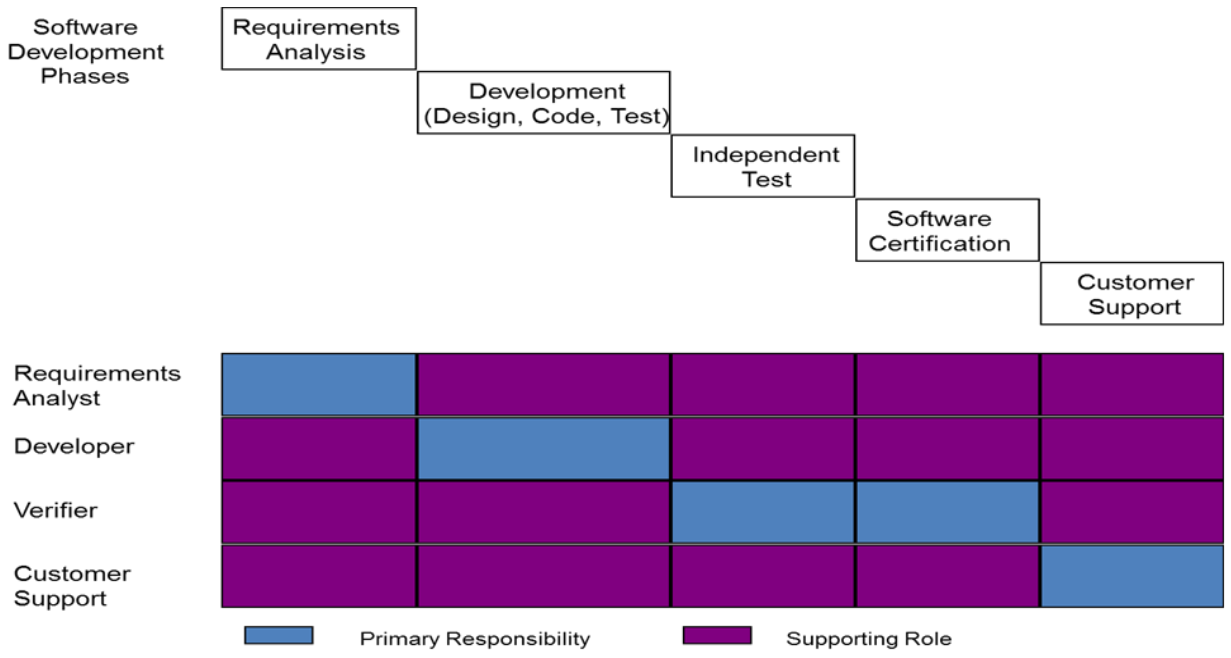


Figure V-1 PASS Roles and Responsibilities

Figure V-2 shows the approach to testing that was applied prior to STS-1 flight. Figure V-3 shows those levels of testing that have continued to be applied across all post STS-1 development and flight release testing.

The levels of testing that were discontinued after STS-1 were Level 4 (System Interfaces / Mission Profiles) and Level 5 (Mass Memory Utilization, Tape/Listing Validation, and System Level Testing). These were specific tests to insure that the PASS system operated properly at system interfaces, supported the phase implementation of flight capabilities (entry only, entry/ascent, and finally entry/ascent/orbit), and verified deliverable products. After STS-1, these activities were either no longer required or else integrated into the Level 3 testing. System Interface
Testing was done on an exception basis as needed by delivery of early releases (prior to Configuration Inspection (delivery to NASA)) to the Software Avionics Integration Laboratory (SAIL) which included a full complement of flight or flight equivalent avionics hardware.

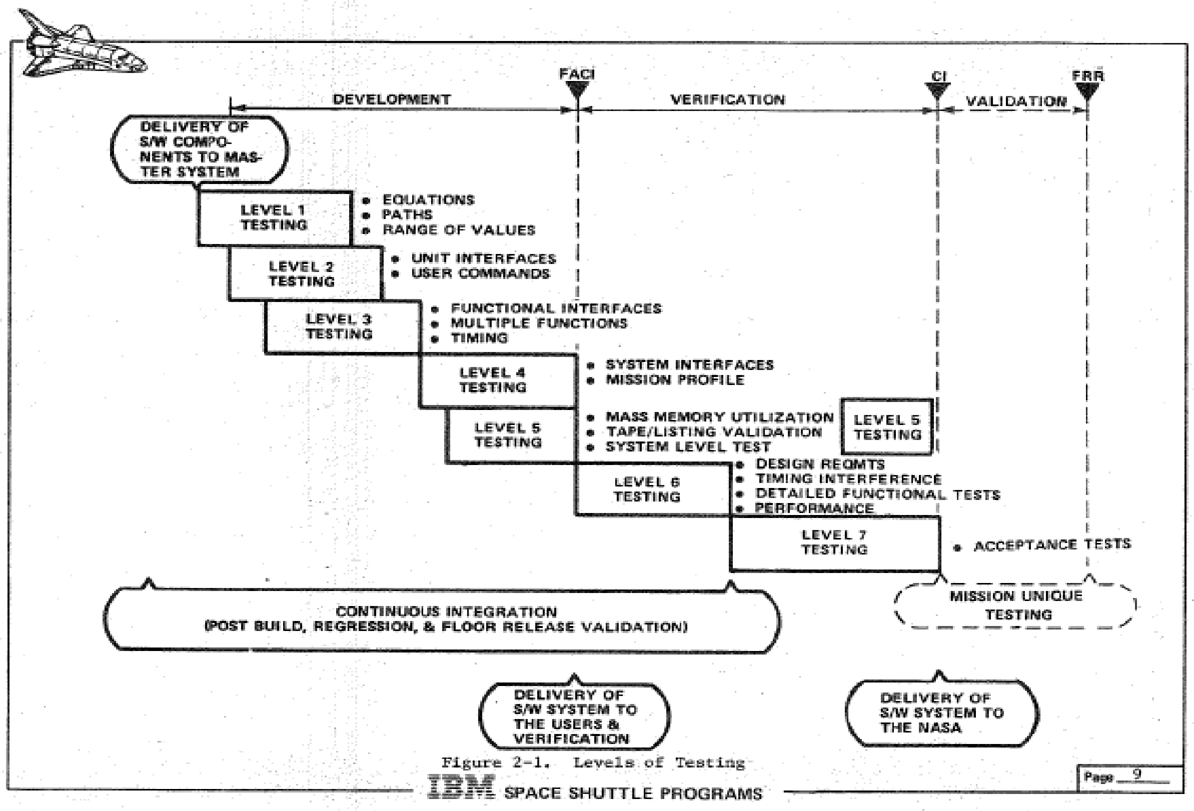


Figure V-2 Initial PASS Testing Approach

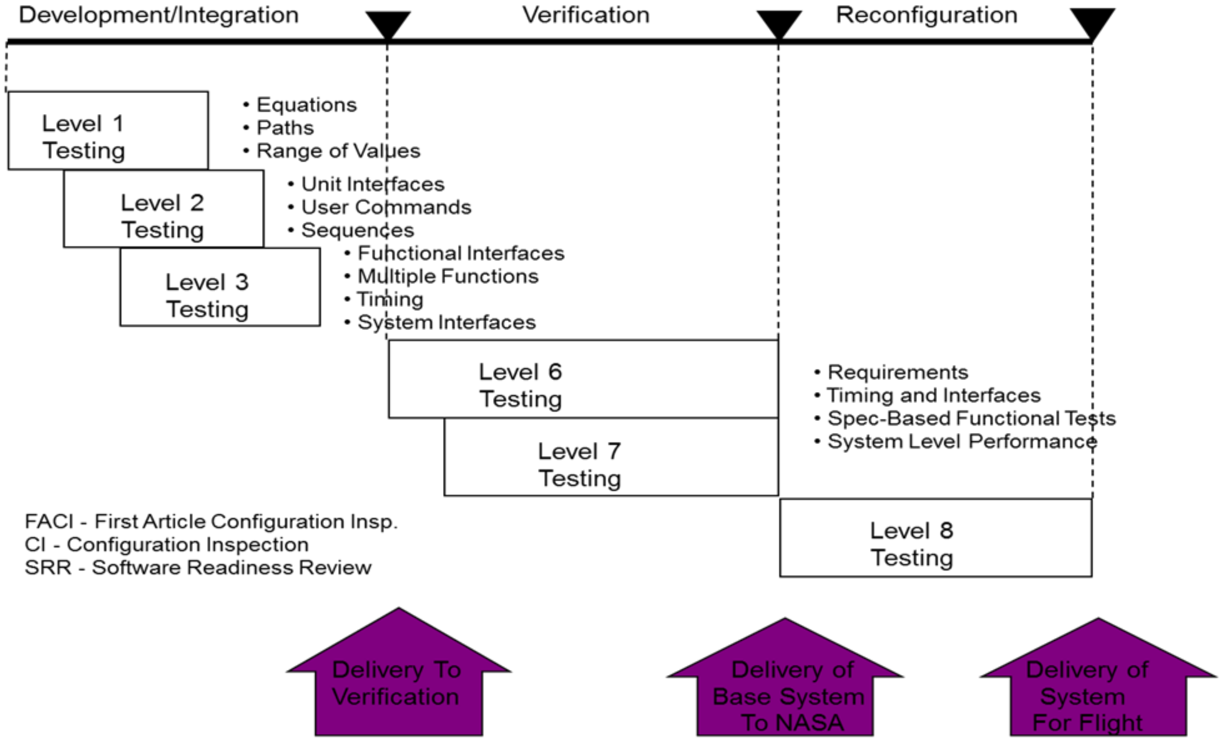


Figure V-3 Final PASS Testing Approach

The PASS development life cycle began with Systems Engineers and Requirements Analysts developing or reviewing requirements. The System Software requirements were developed jointly between NASA and the PASS organization. The majority of the PASS application requirements were generated by "Mode Teams" (what today would be called Integrated Product Teams) in which the PASS Requirement Analyst was an active member along with NASA and multiple companies, including Rockwell International.

Regardless of the organization that generated the requirements, all were submitted to the NASA Shuttle Avionics Software Control Board (SASCB). Requirements were then inspected, defects identified, defects corrected, re-inspected if necessary, and dispositioned (approve, disapprove or defer) by the SASCB. The overall goals of the PASS Requirements Evaluation process were:

- Assess and improve the correctness/quality of the requirements change
- Insure that all interfaces were consistent
- Insure there were appropriate Crew overrides and inhibits
- Insure that the requirements were:
    - Capable of achieving the intent of the change
    - Consistent with system constraints
    - Unambiguous
    - Feasible
    - Testable
    - Consistent with safety requirements
    - Fail operational / fail safe within affected subsystems

The requirements evaluation process required the implementation approach be determined and detailed size/effort estimates complete for code development, code testing, and if there were any tool impacts to the Software Development Facility. The process also identified any other impacts such as Mission Control Center (MCC) or Launch Processing System (LPS) changes, procedures or training changes, or changes to flight design requirements.

Every function impacted by the Software Change Request (SCR) was evaluated by the PASS Requirements Analyst (RA), Developer, and Verifier assigned to that function.

- The RA was the in-house expert (initially) on the function requirements and the change, and the point of contact between PASS and the technical community for the function
    - This was a non-programmer role with background to assess whether the proposed change could achieve the stated intent
- All reviewed the proposed change to identify issues
- Issues were tracked in a database
- The RA worked with the SCR author and the community to resolve issues and track issues to closure
- The Developer and Verifier estimated cost
- All provided inputs to USA risk assessment for the SCR

After STS-5, requested changes were targeted to specific Operational Increment (OI) releases. The process began with a list of potential candidate changes proposed by various organizations or individuals within the NASA or Space Shuttle contractor community. The SASCB ranked the changes based on program benefits including (a) Safety upgrades, (b) Performance enhancements, and/or (c) Cost savings. Based on available resources, a subset of potential candidates were approved for requirements development and placed on the candidate list. A change on the candidate list was brought forward to the SASCB if it was ready for approval including all major issues resolved and impacts /risks identified. The change was added to the OI baseline if the SASCB agreed to the change and it fit within the available resources.

The next phase of the development cycle was the actual implementation of a baselined change. All implementation work on a change was controlled through the configuration management system which identified (a) approval status of the change, (b) affected requirements functions, (c) code modules to be changed, and (d) builds (e.g., OI and flight release) for which the changed code was scheduled. Once authorized, the Developer made changes to the design documentation and the code. The Requirements Analyst and Verifier also made updates, if needed, to other maintenance documentation used to aid traceability.

Design and Code Inspections were the central defect detection mechanism in the PASS development process. Design and Code Inspections were mandatory for all code changes, with mandatory participants of (a) Moderator, (b) Developer, (c) Developer Peer, (d) RA and (e) Verifier. All reviewers checked compliance of design/code to requirements. The RA and Verifier would bring independent understanding of requirements as well as knowledge about the operational use of the software and the planned verification and validation testing. Peer and Verifier also reviewed for compliance with programming standards. Each reviewer was required to fill out a checklist intended to guide the pre-inspection review. Actions identified during the inspection were recorded in a design/code actions tracking database. The moderator tracked the actions to closure before closing out the inspection. The moderator and team used documented criteria to determine the need for a re-inspection.

Development Testing was essentially a unit test of the changed module integrated into the rest of the FSW. Over time, Development Testing evolved from the Level 1 (Equations, Paths, Range of values – Unit Test) and Level 2 (Unit Interfaces, User Commands, Sequences – Subsystem Test) into a scenario based test of the unit fully integrated into the rest of the FSW due to the ease of producing an integrated system during nightly development mini-builds. Development Testing was conducted by Developers, but typically not the programmer of the module under test. The Verifier did not participate in this process to avoid corruption. The Requirement Analyst participated in the pre-test scenario inspection to help ensure requirements coverage, but was not required to review results. During the pre-STS-1 period, the Requirement Analysts wrote the test procedures for Development Testing and maintained off line bench programs that produced expected results for the test procedure.

Once all of the planned inputs had completed development, inspection, and development testing, and all issues had been resolved, the set of modules were ready to be promoted to the baseline code library for a specific system or multiple systems. The Build Coordinator checked for a complete set of inputs for the scheduled content including electronics checking that the submitted modules exactly match the version that was inspected and that all major actions were closed or waivers approved by management. Once all inputs had been checked, the build was initiated and build outputs were checked for error-free completion.   Static analysis tools were run to verify correctness of the integrated build outputs.

The Level 3 test group performed software integration testing immediately after each build by executing a set of standard System Integrity Tests (SITs) that exercise the major capabilities for the phase. In addition to providing a gross confirmation that the new system cycles and “hangs together”, Level 3 produced a set of restart checkpoints that were used by subsequent phases of testing to establish initial conditions.

Detailed Verification (Level 6 Tests) was “Independent” in the sense of a separate management structure from the development group, reporting directly to the PASS director. Verification activities were primarily performed by the “Level 6” or “Detailed” Verification group who were non-programmers familiar with the requirements and code within their responsibility.   Level 6 Verification was responsible for demonstrating that every new, changed, or deleted requirement was correctly implemented to insure that the code exactly met requirements. Level 6 initially produced a set of Verification Test Procedures (VTPs) that establish the test conditions, test steps, and expected results. These were reviewed by the NASA customer.   After the VTPs were approved, a set of Verification Test Cases (test scripts) were developed, debugged, and executed and the results were reviewed with the NASA customer to insure that all expected results were achieved and no unexpected results or error conditions were observed. The Level 6 organization was also responsible for executing a delta-code process where every added, changed, or deleted line of executable code must be mapped to a test case (or alibied) to insure complete coverage of the changes made by the Development organization.

Validation activities were primarily performed by the “Level 7” or “Performance” Verification group which mostly involved testing in the Guidance, Navigation, and Control (GNC) and System Software (SSW) application areas. The Level 7 test group evaluated every change for a potential FSW or GNC performance impact.   Changes with an expected or intended performance affect were candidates for Level 7 capability tests that were tests specifically designed to demonstrate and evaluate the performance.   For GNC capabilities, the performance evaluation insured that the vehicle flew as expected and that all physical constraints (e.g., thermal, structural) were maintained after implementation of the change. From a SSW standpoint, performance factors such as CPU loading and I/O timing were evaluated. In addition to capability tests, Level 7 executed and analyzed a series of SITs that provide coverage of all operational phases to insure adequate performance.

All PASS testing was performed on the Software Development Facility (SDF). This facility had major testing capabilities and utilized flight hardware to execute the flight software under test. This facility used a high fidelity, closed loop simulator which could test with up to 3 flight-equivalent General Purpose Computers (GPCs) in various combinations (GNC, SM, BFS). Test cases were driven by a very powerful test script language. The facility used features of GPC hardware and HAL/S to stop a flight-equivalent processor and flight-configuration code at arbitrary points and initiate event-dependent inputs and test scripts based on logical combinations of conditions in the GPC and/or environment or collect essentially any data desired from the GPC and/or environment. The facility supported checkpoint/restart of a flight-configuration system.

Performance Verification (Level 7 Tests) attempted to show the software performed as intended by exercising the software in the applicable scenarios, and scenarios that might be affected, and evaluating the system performance against the desired improvement (propellant saving, etc.). Performance Verification evaluated the system performance against system or vehicle constraints and accepted "good behavior". Performance Verification performed a set of nominal and off-nominal full mission phase regression tests to demonstrate overall integrity of the modified system. Level 7 test plans were reviewed with the technical community (particularly the GNC and Flight Techniques panels) and approved by the SASCB under a special SCR. Test results were reviewed at the interested working groups, typically a GNC panel.

The PASS and associated documentation and support tools were delivered to NASA at the Configuration Inspection (CI) milestone. The CI milestone signified PASS development and verification was complete, and software was delivered to NASA and ready to reconfigure for crew training and full integration testing. It was at this point that the Contractor certified that all required processes had been fully executed and the resultant software was, to the best of their knowledge, error free.

After the CI, the Reconfiguration process was executed to bring the flight specific data together with the newly released software load to produce an integrated flight load. Each OI typically supported about six to eight flights. Once each flight load was produced, PASS Level 8 testing performed regression testing similar to the Level 7 tests, but with software reconfigured with flight-specific data. The Level 8 group tested the final mission specific flight software release after the release had been reconfigured for the mission specific payloads and flight profiles.

The Integrated Avionics Verification (IAV) group conducted full system integration testing at the SAIL. SAIL could be configured with essentially a full complement of avionics. SAIL was configuration tracked and maintained as a member of the Orbiter fleet (OV-095). SAIL was a real-time, high fidelity simulator with a full cockpit which focused on human-in-the-loop testing and testing interfaces between GPCs and other avionics systems (e.g., Multi Purpose Electronic Display System - MEDS). SAIL had extensive hardware interface testing capabilities not available in the SDF including interface testing between the flight and ground systems (MCC, LPS).

Critical Tools were defined as Software Development Facility tools which directly modify the content of the flight software or tools which were used to verify the flight software. All critical tools were configuration managed and developed in a manner equivalent to that used for the PASS flight software. Critical tools defects were analyzed and were required to be closed during the process leading up to certifying that the PASS flight software was ready to support a Space Shuttle Mission.

Project Office / Project Coordination (Project Management) served the function of a Project Management Office (PMO). The Project Office was part of the organization structure during the development of major software releases for STS-1, STS-2 and STS-5. The Project Office focused on both baseline configuration management as well as contract modification requests. With the transition to Operational Increment development with candidate lists, the need for contract modification requests was greatly reduced. The Project Coordination organization evolved to focus on both baseline configuration management and process adherence. During this period a set of Principles of Providing High Reliability Software evolved which included:

- Safety certification based on process adherence rather than product.
- Assumption that a known, controlled, repeatable process will result in a product of known quality.
- Use of "trusted" tools to develop, build, release and maintain the software.
- Use of measurements to continuously assess the health of both the process and the product.
- Relationship between quality and reliability must be established for each software version and statistically demonstrated for the required operational profiles.

- Quality must be built into the software, at a known level, rather than adding the quality after development.

The Project Coordination organization reported directly to the PASS Project Manager and was organized separately from the other organization roles. This organization structure provided an independent matrix control to insure process adherence.

At the time the software was released by the Development organization to the Verification organization via the build process, the software became subject to the Discrepancy Report (DR) process for the remainder of its lifetime. Every Discrepancy between the coded software and approved requirements was documented as a DR. The DR process was an extremely rigorous process executed by the Project Coordination organization to facilitate the documentation, analysis, correction, and root cause analysis. Each DR underwent an extensive root cause analysis that identified all process and training causes and identified corrective actions. Once the causes were identified, the rest of the FSW was evaluated to determine if any additional discrepancies could have resulted from the same causes. It should be noted that DRs were given high visibility in the PASS Organization and that the historical error rate for DR fixes was extremely low.

VI. PASS History of Development and Operations

The organization has existed in four companies: (1) IBM Corporation from March 10, 1973 through December 31, 1993, (2) Loral Corporation from January 1, 1994 to April 21, 1996, (3) Lockheed Martin from April 22, 1996 to July 3, 1998, and (4) United Space Alliance from July 4, 1998 until August 12, 2011.

Early in the development of the Space Shuttle, NASA managers made a decision that the PASS would be developed by IBM under a separate contract. There were several reasons for this. First, NASA felt that Johnson Space Center had the software management expertise to handle the contract directly based on their experience during the Apollo Program. NASA viewed software as the most critical component of the Shuttle, as it ties the other components together. NASA also felt that the closeness of contractor to Johnson Space Center also facilitated the ability of NASA to manage the project. One of the lessons learned from monitoring the developer in the Apollo era was that by having the software development at a remote site, the synergism of informally exchanged ideas is lost. NASA and its contractors made over 2,000 requirements changes between 1975 and the first flight in 1981.

The period of early 1970's to 1982 was characterized by major technical challenges in terms of infrastructure, programming languages, and requirements definition. One area of challenge was in terms of memory and CPU speed limitations of AP-101B General Purpose Computer. Most of the processes followed over nearly 40 years were in place by the late 1970s as the infrastructure matured, especially the simulation capability in the Software Development Facility (SDF) with high fidelity flight equivalent General Purpose Computers. The overall capacity for execution jobs in the SDF was limited until after 1986. During this time, the Design/Code inspection was conducted by the Development Organization including Developer, Requirements Analyst, and Peer Programmer. Unfortunately, there are no measurements on inspections available from this period. The rigorous testing program including 7 levels of testing prior to Configuration Inspection (CI) and Integrated Avionics Verification in SAIL after each release was in place. In addition, several other facilities were used for software verification. There were 24 Interim releases provided to field users prior to STS-1 over a 2 year period. During this time 2764 Process Discrepancy Reports (DRs) found prior to Software Readiness Review (SRR) for STS-1 for an overall verification effectiveness of over 90% of all introduced DRs were found prior to STS-1. All DRs have been analyzed to determine exactly which release the DR was first introduced. DRs introduced prior to STS-1 continued to be found in 2011, although the discovery rate dropped to about on DR per year. For over one year prior to STS-1, the compiled and linked system was frozen and all mandatory changes were made using machine language patches. In parallel, these same mandatory changes were implemented on the STS-2 system as high order language source updates. In an analysis of the quality of STS-1 versus STS-2, it was determined that the quality of the machine languages patches for STS-1 were higher than the corresponding source changes on STS-2. Causal analysis determined that the significant difference was that Verification Analysts were added to the patch inspections due to the perceived high risk of making machine language patches. This immediately led to a process change where the Verification Analyst was a required Design/Code inspection participant on all changes effective with the release for STS-5 and subsequent.

Just prior to STS-2, during crew training, an error was discovered when all three Space Shuttle Main Engines (SSME) were failed during a training session. This resulted in PASS stopping commanding to all displays and the crew engaging the Backup Flight Software (BFS). This failure of the PASS was the first product DR which, based on PASS only (no BFS), would have resulted in loss of crew and vehicle. This type of DR was classified as a Severity 1 DR. Note that in this case, there were optimum indications to the crew to engage BFS to safely control the Space Shuttle.

Analysis of the training scenario revealed that a necessary element of the failure was the very specific timing of the three SSME failures in relation to sequencing to connect the Reaction Control System (RCS) jets to an alternate fuel path. The corrective and preventative actions from this error led to the development of an analysis technique called Multi-Pass Analysis which has been a major contributor to the long term reliability of the PASS.

A Multi-Pass function requires more than one pass (execution of software code segment) to complete its defined task. It typically has distinct stages, and control flow in each pass through the code that are dependent on state data from the previous pass along with external inputs that may change over time as the task executes. Multi-Pass functions needed to be evaluated for (a) data that was assumed to be static unexpectedly changes, (b) state data used in decision blocks takes on an unexpected value, (c) Multi-Pass function was requested to restart prior to completion, (d) Multi-pass function was terminated before completion, and (e) Multi-pass function was restarted after normal or abnormal completion.

One difficulty in evaluating Multi-Pass functions was defining all scenarios for which the Multi-Pass function was expected to perform, particularly those where unique timing of interrelated events was involved. Failure to consider all scenarios could result in the Multi-Pass function producing undesired results when safety critical hardware was being controlled.

“Multi-Pass” analysis was a method that allows all possible scenarios to be identified simply from analysis of the software code. All variables that were used in decision blocks within the Multi-Pass function were identified. The software code was then analyzed for each decision variable to determine if the variable could be changed while the Multi-Pass function executed. If the variable could be changed, then the conditions when changes were permitted to occur were identified. Based on the Multi-Pass function code structure and the conditions when decision variables could be changed, it was possible to define all theoretical scenarios. These scenarios were then analyzed to determine which of these scenarios the software should support. Rejected scenarios were not analyzed any further. All supported scenarios were then evaluated to determine how the Multi-Pass function software code responded and if the response was acceptable. When necessary, requirements were changed to insure an acceptable response.

In summary, the essential elements of “Multi-Pass” analysis were:

- Identify all theoretical possible scenarios through structured analysis of design/code including any implementation unique scenarios
- Identify the subset of theoretical possible scenarios that are to be supported
- Insure that the Multi-Pass function produces acceptable responses for all supported scenarios.

The period from 1983 to 1985 was characterized by ongoing challenges in terms of adding functions and maintaining the system in the face of memory and CPU speed limitations of AP-101B GPCs. This resulted in a discrepancy which could have resulted in failure to separate the External Tank. Significant IBM analysis effort was expended in analyzing module-by-module timing data to identify potential solutions. IBM designed, tested, and delivered a 12 half-word code patch to ensure a sufficient timing delay between the PASS-computed commands and the output of those commands to the hardware over a 48 hour period prior to STS-41D launch on 08/30/1984.

The Systems Management / Payload software was redesigned for STS-5 due to both memory and CPU issues encountered when adding payload support.

The Pre-Build Design/Code inspection conducted by the FSW Organization now included Developer, Requirements Analyst, Verification Analyst, and Peer Programmer. Measurements on inspections became available starting in this period and showed the inspection process effectiveness in identifying errors rapidly rising.

The transition was completed from manually generated vehicle and payload flight specific code to code generated by automated pre-processors from reconfiguration databases.

As major development was completed with STS-5, de-staffing was initiated by IBM in 1985 via placement on other IBM projects. Process changes in this time included transitioning from long development time for releases into frequent Operational Increments with delta time between Configuration Inspections (CI) on the order of four months. The net effect was reduced verification time per release. Characteristics of the period included a significant number of Product DR's introduced in this period which were discovered in flight, including three Product DRs which affected mission objectives that were patched during flight. Additional Released Severity 1 DRs were discovered, creating concerns to (a) avoid future introduction and (b) find any remaining existing Severity 1 DRs. The program desire to become fully operational resulted in continued high demand for software CR changes with some risk of over-commitment. One manifestation of this was increasing late change traffic on OI's (Over 50 % of the OI-7C content baselined post FACI).

The period of 1986 to 1988 was characterized by the accident in which Challenger was lost and the subsequent focus on return to flight with a significantly safer Space Shuttle including the flight software. Space Shuttle Challenger was lost with its crew on 01/28/1986. This resulted in a stand down until STS-26 flew on 09/28/1988.

This time period focused on the actions taken to achieve the return-to-flight on STS-26. Actions included rigorous review of software requirements; numerous safety changes were identified and implemented on OI-8A and OI-8B. The NASA Program Review Board (PRB) assigned IBM an action to compute the probability of the loss of a shuttle and crew due to a PASS FSW error, while ignoring the potential for the Backup Flight System (BFS) to safely engage. The resulting answer was that the risk of loss of the Space Shuttle due to a PASS failure was approximately 1 in 1,600. While executing tasks to safely return the shuttle to flight, eight PASS Severity 1 DRs were discovered during this period in addition to two found in 1985. However, after this period, IBM virtually eliminated released Severity 1 DRs.

This was a very, very busy period, especially in 1988. Activities in work included: (a) Completing special studies under the label "Revalidation", (b) Preparing for STS-26 flight including expanded Flight Readiness Review (FRR) Process, (c) Completing verification of OI-8C and development of OI-8D, and (d) Preparing to resume transition to the AP-101S upgraded computer which gave significant increase in available General Purpose Computer CPU and memory.

Transition to the AP-101S upgrade flight computer started prior to the Challenger accident (AP-101S required operating system changes). Development work on an AP-101S flight system was abandoned (OI-9, OI-10, OI-11). Return-to-flight flight mandatory changes were implemented on AP-101B systems (OI-8A, OI-8B, OI-8C, and OI-8D). Only after we had resumed flying did we re-implement mandatory AP-101S system software changes on OI-8F (started at the end of this period).

Significant changes were made to the ability to execute tests in the Software Development Facility (SDF) / Software Production Facility (SPF). At the start of this period, there was one Flight Equipment Interface Device (FEID) that could run multi-computer runs by itself, and three FEIDs that could run single computer runs or be combined to run multi-computer runs. At the end of this period, there were six FEIDs that could each run multi-computer runs. This resulted in an increase in capacity to run test cases in the SDF and SPF by at least a factor of 3. This increased test capacity was a significant contributor to a reduction in in-flight DRs compared to product DRs found on the ground in later periods.

The period from 1989 to 1993 was bounded by the STS-26 return-to-flight launch on 09/28/1988 at the beginning and ending with the sale of IBM Federal Systems Division to Loral Corp. effective January 1, 1994. Also at the end of this period, the IBM Federal Systems Division Houston contract on Space Station Freedom software was terminated. Quality of new development was maintained over this entire period at record low levels approaching 0.1 DR/KSLOC Product Error Rate. Available AP-101S memory and CPU speed resulted in major capability additions. Achievements in quality were recognized in 1989 when NASA used the PASS project for a "practice" CMM assessment and concluded the PASS organization to be at CMM Level 5. This was the highest possible rating and the first time it was achieved. The Software Engineering Institute was shocked at this possibility and sent a SEI team to validate the NASA team findings. This started an on-going collaboration with the Software Engineering Institute which lasted for nearly 20 years.

IBM negotiated a five year sole source extension of the contract to support PASS FSW development and maintenance starting in July, 1993. The contract included provisions for gradually reducing the staffing level over the five years.

Late 1993 was not a good period for IBM Federal Systems in Houston. While IBM FSD Houston was re-planning / transitioning from the Space Station Freedom Program to the International Space Station Program, IBM's contract work on Space Station software ended. In parallel, IBM commercial divisions were struggling with revenues and profits as the mainframe era came to an end and the PC/server era evolved. To raise cash, IBM made a strategic decision to sell its space and defense businesses.

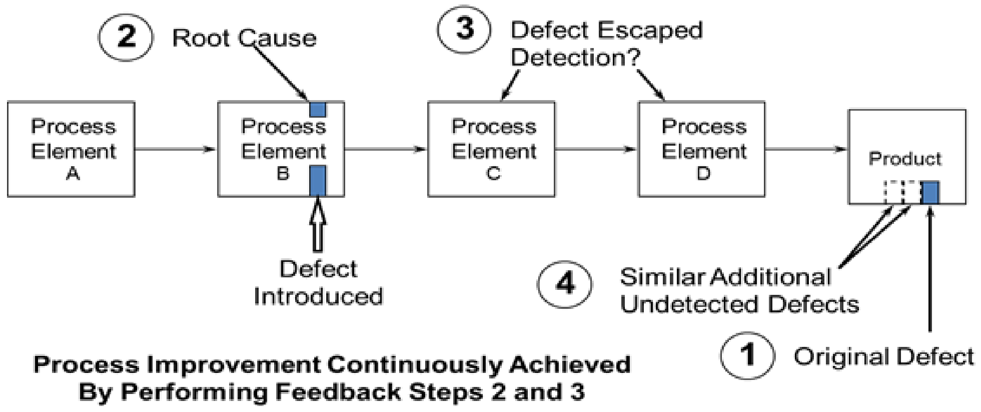
Some IBM Houston personnel were scattered as the remaining Space Shuttle work sold to Loral Corporation. Many IBM Houston personnel either elected early retirement packages, transferred to other IBM projects and divisions, or voluntarily left for more promising job prospects outside of Loral. Leadership immediately following the transition to Loral was a major positive. The initial Loral executive manager was Mike Coats. Tom Peterson, as PASS program manager, provided significant stability in this period.

However, it still was a traumatic period as 1993 ended. The Space Shuttle PASS project lost virtually all personnel with less than four years experience. Other experienced personnel left the project. IBM and former Ford Aerospace subcontractor personnel were merged into one new Loral organization.

The period from 1994 to 1997 covered the time from transition from IBM to the time the project transitioned to United Space Alliance. IBM Federal Systems Division was sold to Loral Corporation as of January 1, 1994. On April 22, 1996, Lockheed Martin completed the acquisition of Loral Corporation's defense electronics and system integration businesses including the former IBM Federal Systems Division. United Space Alliance (USA) and NASA signed the Space Flight Operations Contract in September 1996 for USA to be the single prime contractor for the Space Shuttle Program. NASA's intention was to transfer the PASS FSW contract work to USA at the completion of the five year contract signed in 1993. PASS FSW Contract work transitioned to USA on July 4, 1998.

Morale within the PASS FSW project began to deteriorate after the loss of Mike Coats in 1996 when he became Vice President of Civil Space Programs for Lockheed Martin Missiles and Space in Sunnyvale, California. Corporate level process improvement activities affecting the PASS Space Shuttle project became less focused after the Houston organization was re-organized separate from other parts of the former IBM Federal Systems Division. Some personnel were extremely distracted throughout this period.

During this period, a major process escape occurred. Due to a combination of over commitment and personnel distraction, one small change out of many was released with multiple errors. The subtle effects of one of these errors were detected during analysis of post flight data from STS-79. The Defect Prevention Process implemented in the early 1980's immediately identified this major process escape.



2 Root Cause
3 Defect Escaped Detection?
Process Element A
Process Element B
Process Element C
Process Element D
Product
Defect Introduced
4 Similar Additional Undetected Defects
1 Original Defect
Process Improvement Continuously Achieved By Performing Feedback Steps 2 and 3

Figure VI-1 PASS Root Cause Analysis Process

The whole Defect Prevention Process was carefully structured and managed NOT to be a blame game. PASS processes were designed so that no individual should be a single-point failure. Individual responsibility was expected and monitored, but every activity should be cross-checked by at least one other person. Every defect represented an opportunity to improve a process. Suggestions for process improvements were fed back to the individual processes.

The errors detected on STS-79 were the result of several personnel performance failures. Major process changes were implemented which included detailed management review of the participants in all inspection processes, and major additions to the inspection process for Verification Test Cases results.

The period from 1998 to 2002 spans from transition of contract work to United Space Alliance on July 4, 1998 until the second shuttle accident involving loss of crew and vehicle (STS-107) on February 1, 2003. Early 1998 was difficult as the time to transition to USA approached. NASA and United Space Alliance did everything in their power to make the transition smooth and as seamless as possible to employees. Once the contract transition was completed, and employees were part of USA, there was a vast improvement in morale. Employees were well treated by USA, which tied into improvements in software release quality.

NASA was focused on extending the life of the Space Shuttles to 2020. Several major upgrades were in the process of being implemented including the Cockpit Avionics Upgrade (CAU). In 2002, PASS FSW development resources began work on OI-41 which was to support the PASS changes necessary for CAU. CAU was a very large, major development activity with USA as prime for the development of hardware, software and integration. Major SAIL facility modifications were also required along with major PASS FSW and support software (Application Tools) changes. Morale and product quality were at all time highs.

The period of 2003 to 2005 was characterized by the Space Shuttle Columbia accident and actions taken while the Space Shuttle Program completed changes for return to flight.

The organization was saddened when the Space Shuttle Columbia and crew were lost on February 1, 2003. During the return to flight activities, OI development in this period was limited to CAU. Non-CAU work was limited to additional flight changes to OI-30 for return to flight. Cockpit Avionics Upgrade was making meaningful progress. President Bush then changed space policy as a result of the Columbia accident on January 14, 2004 when he announced that the Space Shuttle would end by 2010 and a new exploration program Constellation would begin. CAU development terminated very late in 2004 after three years of effort.

The focus of the period from 2006 to 2008 was on flying shuttle missions and maintaining the critical skills to provide mission support and to resolve any issues in a timely manner. Three OI's were developed in this period (OI-32, OI-33, and OI-34). To insure continued process quality and efficiency, United Space Alliance Flight Software Element completed a CMMI appraisal in November 2006; assessed at CMMI Level 5.

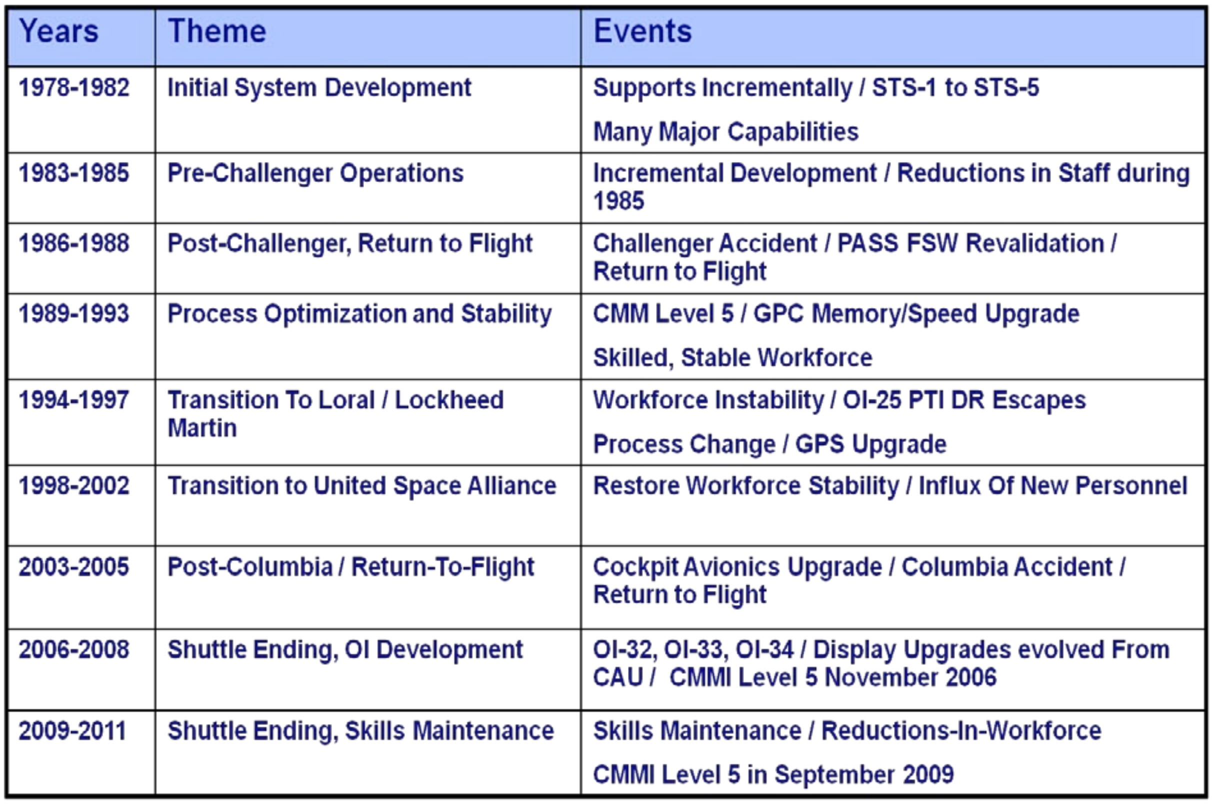
Generally, content size was getting smaller and smaller. OI implementation of change instruments was sometimes assigned across multiple teams just to spread the exposure to code and process. After return-to-flight, there were a number of Space Shuttle Program level technical issues that constrained the flight rate during this entire period as solutions were found to the technical issues.

The period of 2009 to 2011 was characterized by the phrase "Complete the Mission" as the Space Shuttle Program end targeted for October, 2010. Eight shuttle flights in 14 months provided a focus which countered the uncertainty of the approaching end of shuttle through STS-132 (May 2010). There was a focus on executing training activities to maintain critical skills in place of production OI work in prior periods. To insure continued process quality and efficiency, United Space Alliance Flight Software Element completed a second CMMI appraisal in September 2009, and was assessed at CMMI Level 5.

The Obama administration announced new space policy on January 27, 2010 which would extend the International Space Station operations through at least 2020, but abandoned NASA's current plans to return U.S. astronauts to the moon.

Payload issues and ISS traffic constraints resulted in slipping the last space shuttle flight ending in March 2011. One final flight (STS-135) was added to the baseline, with the final Space Shuttle flight flown from 07/08/2011 to 07/21/2011.

Figure VI-2 below identifies periods with common characteristics. These common characteristics are typically defined by company changes and major program events including the Challenger and Columbia accidents.



Years	Theme	Events
1978-1982	Initial System Development	Supports Incrementally / STS-1 to STS-5 Many Major Capabilities
1983-1985	Pre-Challenger Operations	Incremental Development / Reductions in Staff during 1985
1986-1988	Post-Challenger, Return to Flight	Challenger Accident / PASS FSW Revalidation / Return to Flight
1989-1993	Process Optimization and Stability	CMM Level 5 / GPC Memory/Speed Upgrade Skilled, Stable Workforce
1994-1997	Transition To Loral / Lockheed Martin	Workforce Instability / OI-25 PTI DR Escapes Process Change / GPS Upgrade
1998-2002	Transition to United Space Alliance	Restore Workforce Stability / Influx Of New Personnel
2003-2005	Post-Columbia / Return-To-Flight	Cockpit Avionics Upgrade / Columbia Accident / Return to Flight
2006-2008	Shuttle Ending, OI Development	OI-32, OI-33, OI-34 / Display Upgrades evolved From CAU / CMMI Level 5 November 2006
2009-2011	Shuttle Ending, Skills Maintenance	Skills Maintenance / Reductions-In-Workforce CMMI Level 5 in September 2009

Figure VI-2 Historical Summary of PASS Organization

VII. PASS Quality

Much of the PASS Legacy will be associated with the extremely high quality of the onboard flight software (PASS) for the Space Shuttle. The PASS organization was a trailblazer in the field of ultra-high quality man-rated FSW. History shows that the quality of the FSW that flew on STS-1 was excellent. Over the next 30 years, the released quality continued to improve. The observed high quality was primarily due to the following:

- **Testing in the most flight –like conditions attainable** - Having real GPCs in the Software Production Facility and an integrated test facility like the Software Avionics Integration Laboratory (SAIL) were extremely valuable both in detecting complex error conditions and in gaining confidence that the system will perform well when lives were on the line.
- **Quality Focused Culture** - Having a workforce culture that believed in zero discrepancies making it to flight. Embedded independent verification and effective peer reviews were very important to producing error free, high quality code.
- **Cooperative Relationship between Government and Contractor** - Having a good, cooperative working relationship between the government and the contractor was essential. Trust between the two allowed for the free flow of lessons learned and process/SW improvements.

From the very beginning, the PASS organization was focused on identifying critical metrics, gathering the data, and closely tracking and analyzing them. Software errors, documented by Inspection Major Errors and Discrepancy Reports (DRs), for PASS were counted in three periods:

1. **Pre-Build errors** - Errors found by Inspection and Development Test Pre-Build (prior to being placed under project configuration control),
2. **Pre-Release Errors (Process DR)** - Process DRs found Post Build until a milestone called Software Readiness Review (SRR) for the first flight off that increment; typically occurs approximately 4 weeks prior to the first flight of a release,
3. **Released Errors (Product DR)** - Product DRs found from SRR of first flight until end of program

A subset of Product DRs are those which occur in either countdown or in flight, called in-flight DRs. There was an additional special category of DRs are called Released Severity 1 DRs. Severity 1 referred to Discrepancies with the potential to cause Loss of Crew and Vehicle (LOCV). These may have been process or product DRs that are released to any field site such as the Shuttle Mission Simulator (SMS), the vehicle at KSC (Kennedy Space Center), or the Shuttle Avionics Integration Lab (SAIL).

Data has been collected and maintained since the beginning of the program for category 2 (Process DRs found Post Build until a milestone called Software Readiness Review), since the flight of STS-1 for category 3 (Product DRs found from SRR of first flight until end of program), and since 1982 for category 1(Errors found by Inspection and Development Test Pre-Build). The attributes of the PASS Configuration Management system and coding standards allowed every single error (DR) document to include information about the system where the error was originally introduced. The source code CM library management system uniquely identified the last update by each individual line of code. Programming standards required a prologue comment that described every line of code changed for each authorizing change control authorization for each update. Once an error (DR) was analyzed to the point that the line of code in error was identified, then the above information allowed the update and change control authorization when the error was introduced to be identified. This information became part of the information about the error (DR) that is recorded in the Configuration Management System.

There are many metrics which demonstrated outstanding performance by the PASS organization. One metric focused on the accuracy with first time correct implementation of error fixes (DRs). This metric was the ratio of new errors (DRs) introduced as a result of attempting to implement a correction to a prior error (DR). For the period from 1998 to 2011, the mean value was 3.7 new Process errors (DRs) introduced for every 100 attempted correction of prior errors (DRs) or 3.7 % with a 95 % confidence interval of 0.5 % to 6.9 %. With verification finding between 85 % and 100 % of errors (DRs) in this period, the results were less than 1 new Product DR for every 200 attempted corrections of prior errors (DRs). This near perfection in error (DR) correction was a major contributor that allowed the PASS reliability to increase by a factor of 16 over the 30 year life of the program.

With a process whose evolution began in the mid-70's when most organizations did not have a clue about how to properly engineer software, IBM was able to create world-class quality and reliability for life-critical Space Shuttle in-flight software operations. The organization and process received multiple honors recognizing 30 years of demonstrated flight safety and ultra-reliability including:

- NASA HQ assessed as SEI Level 5 in 1989 (first ever CMM Level 5)
- ISO 9001 registered process
- CMMI Level 5 in 2006, 2009

Additional Space Shuttle legacy information on quality can be reviewed at:

http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100029536_2010030196.pdf

Additional Space Shuttle legacy Lessons Learned can be reviewed at:

http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100028294_2010031022.pdf

One unique feature of the Space Shuttle PASS quality data was the integrity of the data collection over a 35 year period. This unique set of data allows a look back at prior performance with the full benefit of hindsight and to correlate changes in quality trends with changes in process or culture. The results of that look back is presented here to allow future programs to best assess absolute quality and reliability levels based on early program data.

Data collected on each error (DR) includes which release the error (DR) was introduced on and when the error (DR) was later detected. This information allows the construction of the following plot which shows how many unknown latent defects were present in the PASS flight software over time.

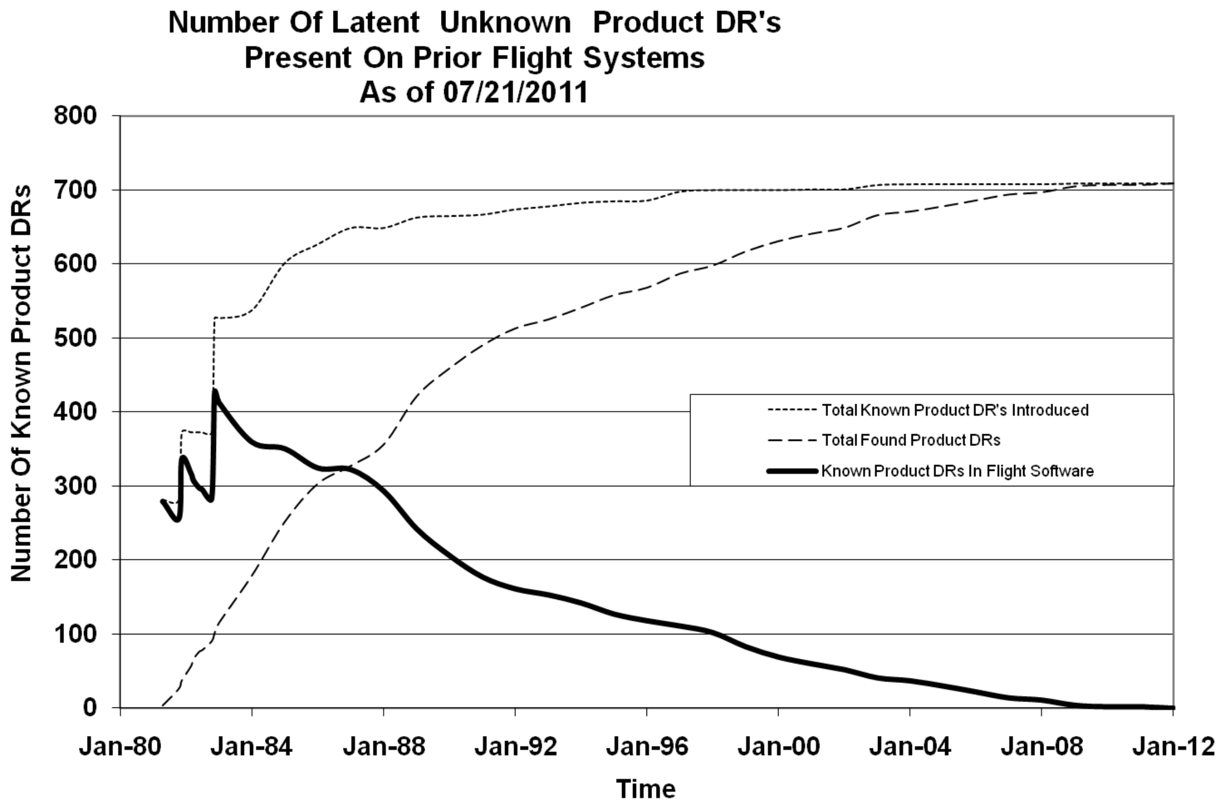


Figure VII-1 Number of Latent Unidentified Product DR's

Figure VII-1 shows that the vast majority of defects were introduced during the three major releases for STS-1, STS-2 and STS-5. Rate of discovery of latent errors (DRs) was relatively constant for almost 10 years and then gradually reduced in proportion to the number of remaining undetected latent errors (DRs). The vast majority of remaining errors (DRs) were either (a) in code that would only be executed in extreme scenarios, (b) in code that was no longer in use for operations, or else (c) was of such insignificant nature that operational behavior was acceptable. Many errors (DRs) were detected during later enhancements during regression testing against the explicit detail of unmodified requirements.

Following the Space Shuttle Challenger accident in 1986, IBM was asked to quantify the probability that a critical error in the PASS flight software could result in a severity 1 condition, e.g., loss of crew and vehicle. Available software reliability models were evaluated and the risk level was conservatively established at 1 in 1000 missions. One innovation applied to the reliability estimation was to model the reliability of each software release independently and then combine all releases for a final system level reliability. In reviewing the available software reliability models, two issues emerged. First, extensive history data on failures was required for each release. With the overall quality of the releases, it was very difficult to have a sufficient number of failures on each release to draw conclusions from. Secondly, the available software reliability models under estimated the risk of initial STS-1, STS-2 and STS-5 releases and over estimated the risk for later releases, many of which have limited failure data to analyze. As a result of these issues, IBM developed an alternate Space Shuttle tailored reliability model. The results of both the alternate Space Shuttle tailored reliability model and the available widely used software reliability models both gave the same conservative 1 in 1000 mission for the PASS to encounter a severity 1 error (DR) in flight. However, the contribution from each release was significantly different from the two models.

As the program approached the end, the analysis was re-performed based on the hindsight of knowing errors that actually happened rather than conservatively predicting what future errors might occur. This re-analysis resulted in an updated reliability estimate of 1 in 1600 mission in 1986. This indicates that the alternate Space Shuttle tailored reliability model level of conservativeness was about a third (estimate 1/1000 versus later improved estimate of 1/1600).

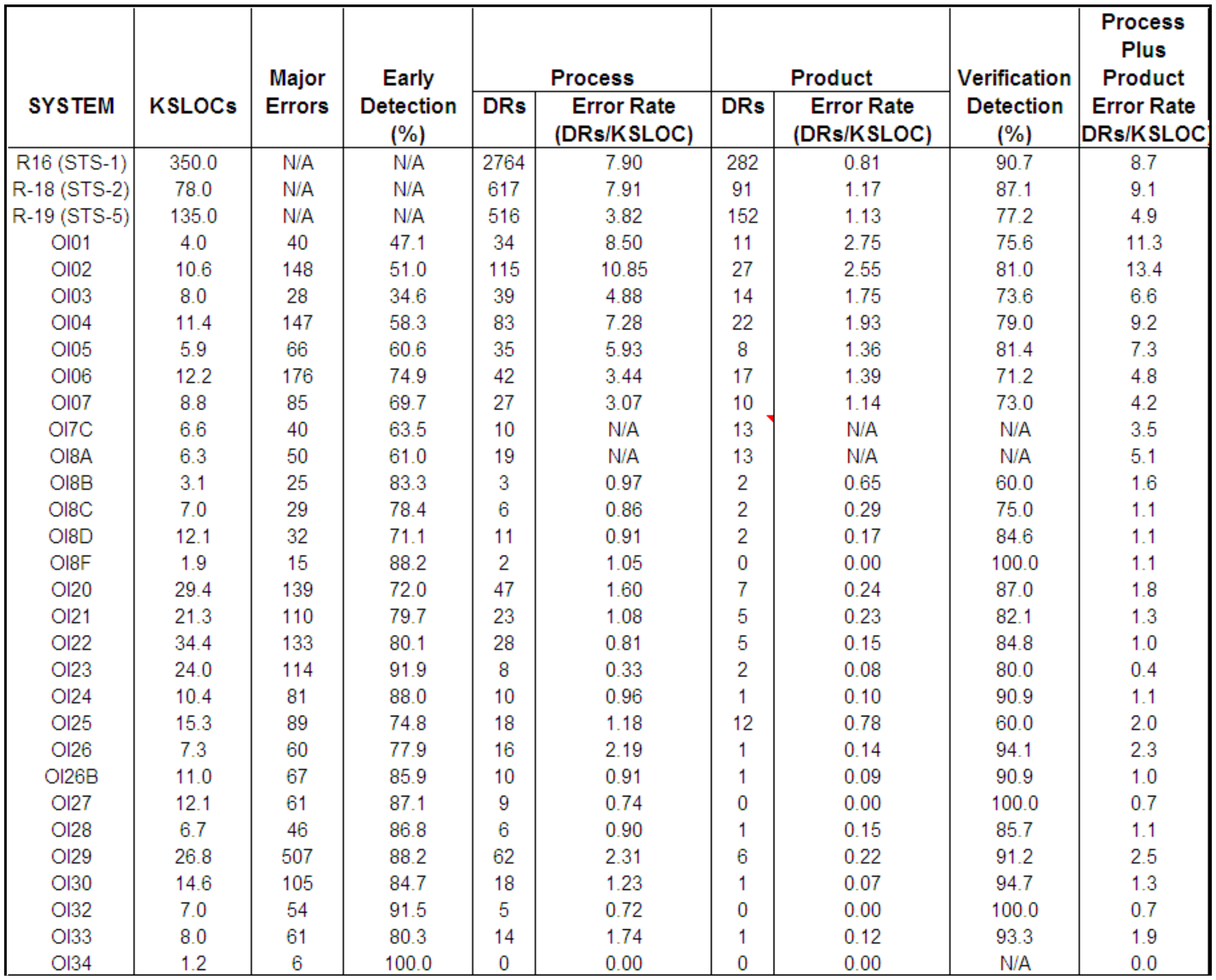
The following table presents key Space Shuttle PASS flight software reliability from three perspectives:

- Mean Time Between Failure in any environment (development, verification, facility use, ground checkout, crew training, and flight)
- Mean Time Between Failure for those failures that occurred in flight including launch countdown
- Number of Shuttle missions until loss of crew/vehicle based on PASS only (does not include the fact that many PASS failures could be recovered by engaging the Backup Flight System (BFS)).

Years	**Calendar Days Between Any Product DR**	**Flight Days Between In-Flight DRs**	**Risk To Shuttle Due To Severity 1 FSW DR**
1978-1982	**6 (STS-1), 7 (STS-5)**	**7 (STS-1), 9 (STS-5)**	**1 in 327 (STS-1) to 1 in 409 (STS-5)**
1983-1985	**10 to 19**	**12 to 24**	**1 in 552 to 1 in 1072**
1986-1988	**29 at STS-26**	**90 at STS-26**	**1 in 1599 at STS-26**
1989-1993	**29 to 42**	**90 to 131**	**1 in 1599 to 1 in 2335**
1994-1997	**42 to 54**	**131 to 120**	**1 in 2335 to 1 in 3161**
1998-2002	**54 to 61**	**120 to 140**	**1 in 3161 to 1 in 3491**
2003-2005	**75 at STS-114**	**235 at STS-114**	**1 in 4212 at STS-114**
2006-2008	**75 to 88**	**235 to 276**	**1 in 4212 to 1 in 4930**
2009-2011	**88 to 94**	**276 to 294**	**1 in 4930 to 1 in 6260**

Figure VII-2 PASS Reliability

The following table contains the basis summary of defect data and product size (as uncommented physical source lines of code) through end of STS-135 on July 21, 2011.



SYSTEM	KSLOCs	Major Errors	Early Detection (%)	Process		Product		Verification Detection (%)	Process Plus Product Error Rate DRs/KSLOC
				DRs	Error Rate (DRs/KSLOC)	DRs	Error Rate (DRs/KSLOC)		
R16 (STS-1)	350.0	N/A	N/A	2764	7.90	282	0.81	90.7	8.7
R-18 (STS-2)	78.0	N/A	N/A	617	7.91	91	1.17	87.1	9.1
R-19 (STS-5)	135.0	N/A	N/A	516	3.82	152	1.13	77.2	4.9
OI01	4.0	40	47.1	34	8.50	11	2.75	75.6	11.3
OI02	10.6	148	51.0	115	10.85	27	2.55	81.0	13.4
OI03	8.0	28	34.6	39	4.88	14	1.75	73.6	6.6
OI04	11.4	147	58.3	83	7.28	22	1.93	79.0	9.2
OI05	5.9	66	60.6	35	5.93	8	1.36	81.4	7.3
OI06	12.2	176	74.9	42	3.44	17	1.39	71.2	4.8
OI07	8.8	85	69.7	27	3.07	10	1.14	73.0	4.2
OI7C	6.6	40	63.5	10	N/A	13	N/A	N/A	3.5
OI8A	6.3	50	61.0	19	N/A	13	N/A	N/A	5.1
OI8B	3.1	25	83.3	3	0.97	2	0.65	60.0	1.6
OI8C	7.0	29	78.4	6	0.86	2	0.29	75.0	1.1
OI8D	12.1	32	71.1	11	0.91	2	0.17	84.6	1.1
OI8F	1.9	15	88.2	2	1.05	0	0.00	100.0	1.1
OI20	29.4	139	72.0	47	1.60	7	0.24	87.0	1.8
OI21	21.3	110	79.7	23	1.08	5	0.23	82.1	1.3
OI22	34.4	133	80.1	28	0.81	5	0.15	84.8	1.0
OI23	24.0	114	91.9	8	0.33	2	0.08	80.0	0.4
OI24	10.4	81	88.0	10	0.96	1	0.10	90.9	1.1
OI25	15.3	89	74.8	18	1.18	12	0.78	60.0	2.0
OI26	7.3	60	77.9	16	2.19	1	0.14	94.1	2.3
OI26B	11.0	67	85.9	10	0.91	1	0.09	90.9	1.0
OI27	12.1	61	87.1	9	0.74	0	0.00	100.0	0.7
OI28	6.7	46	86.8	6	0.90	1	0.15	85.7	1.1
OI29	26.8	507	88.2	62	2.31	6	0.22	91.2	2.5
OI30	14.6	105	84.7	18	1.23	1	0.07	94.7	1.3
OI32	7.0	54	91.5	5	0.72	0	0.00	100.0	0.7
OI33	8.0	61	80.3	14	1.74	1	0.12	93.3	1.9
OI34	1.2	6	100.0	0	0.00	0	0.00	N/A	0.0

Figure VII-3 PASS Error Rates

Key process metrics on error insertion and error detection were analyzed. When comparing these numbers to other programs, it is important to factor in the length of time that the systems have been in operation and that we have continued to track error back to the original system where the error was introduced. This level of data collection and analysis over this time period is extremely rare. The following graph shows how our perception of the STS-1 software release Product Error Rate has evolved over time as more and more errors are found that were introduced pre-STS-1.

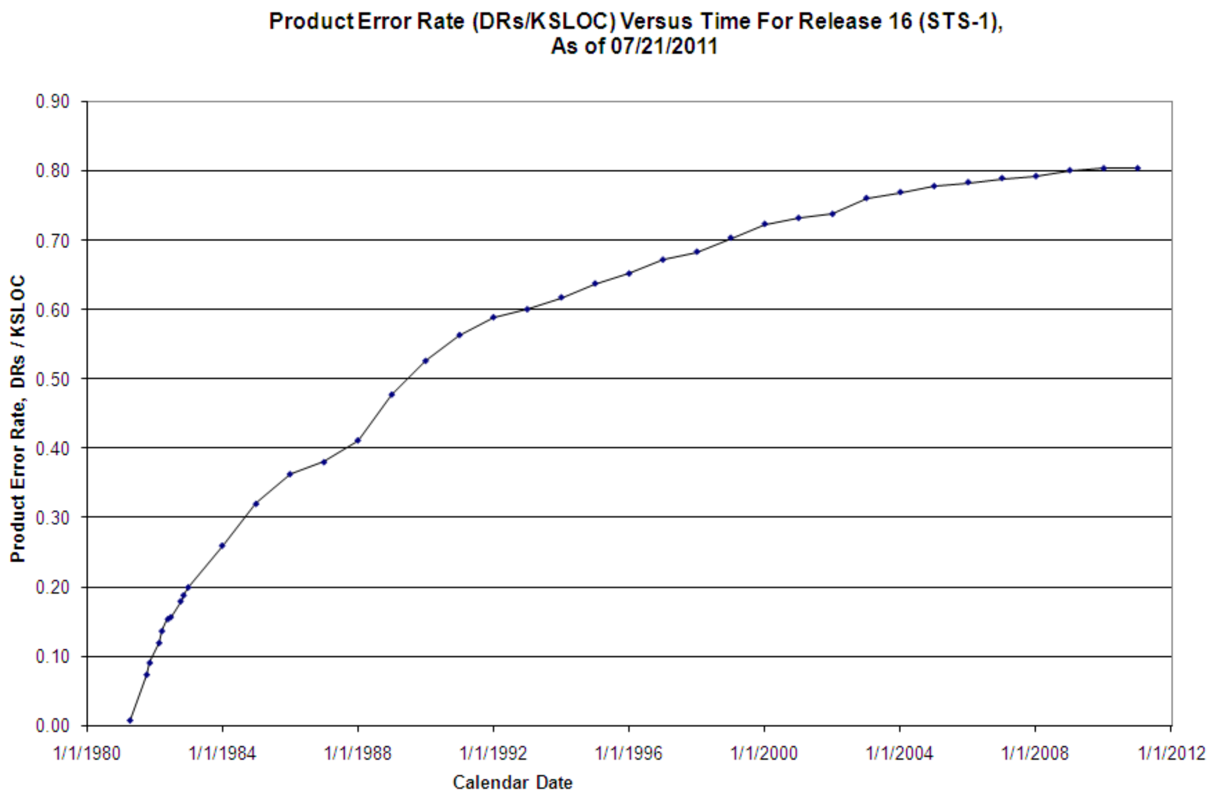


Figure VII-4 Changes in Perception of STS-1 FSW Quality as Operational Data Accumulates

The following table presents key process metrics on error insertion and error detection.

Years	Product Error Rate DRs / KSLOC	Pre-Build Error Detection Effectiveness	Verification Effectiveness (Percent Found By SRR)	Notes
1978-1982	0.8 (STS-1) to 1.1	Information Not Available	77 % to 91 % (STS-1)	
1983-1985	2.8 (OI-1) to 1.1	40 % to 65 %	70 % to 80 %	Very Short Cycle - Release Every 4 Mo.
1986-1988	0.7 to 0.2 (OI-8C)	Near 80 %	60 % to 70 %	Return-to-flight Critical Changes
1989-1993	0.1 to 0.2	80 % to 90 %	80% to 90 %	
1994-1997	0.1 to 0.2 except 0.8 for OI-25	75 % to 85 %	85 % to 100 % except 60 % for OI-25	Isolated Process Escape on OI-25
1998-2002	0.1 to 0.2	85 % to 90 %	85 % to 95 %	
2003-2005	CAU Canceled	CAU Canceled	CAU Canceled	Work on CAU required changes, Later CAU Canceled
2006-2008	0.0 to 0.1	80 % to 100%	95 % to 100 %	
2009-2011	No OI Development	No OI Development	No Development	Reduced Flight System Changes Only, No OI Dev.

Figure VII-5 PASS Error Insertion and Detection Data

Orr's Laws – Sequential Error Detection Processes for Ultra-Reliable Software

From the beginning, the PASS project was focused on metrics and process. The large and robust set of data that PASS gathered and maintained not only served as an engine for PASS process improvements, it also presented many opportunities to draw generic conclusions about best practices for critical software development.

The first of these opportunities occurred in the mid 1980's when Ford Aerospace was a subcontractor to IBM in the role of software developer for PASS. During this period, a subset of the PASS FSW was developed by Ford Aerospace. The Ford developed software went through the same IBM inspection process, including independent verification, as products developed by IBM developers. To maximize the quality of the product coming into the IBM inspection, Ford Aerospace management proposed doing an internal inspection, fixing any errors found, and then submitting the product to the IBM inspection process. The result of data over many years showed that each inspection cycle (Ford internal, IBM with verification) was equally effective of finding the same percentage of errors present at the start of the particular inspection cycle.

The second situation that provided significant insight occurred after a set of process inspection process escapes were identified on OI-25. The inspection moderators proposed new re-inspection criteria that significantly increased the likelihood of requiring a re-inspection if errors might be present. Data from the re-inspection compared to the original inspection also showed that each inspection cycle (original inspection, re-inspection) was equally effective in finding the same percentage of errors present at the start of the particular inspection cycle.

Review of this data and the results of error detection from key testing activities also indicated that *an inspection cycle and a test cycle are equally effective at detecting errors*. The data showed that each inspection cycle (Ford internal, IBM with verification) was equally effective of finding the same percentage of errors present at the start of the particular step (inspection or test). This lead to two general observations ("Orr's laws"):

- The effectiveness of detecting errors remaining at the start of a process step is identical (to a first order) for (a) development only internal inspection, (b) PASS project single inspection, (c) PASS project re-inspection, (d) development pre-build testing, (e) independent verification testing, and (f) SAIL hardware / software integration testing.
- The total program error detection is dependent on the number of error detection sequential steps and the average effectiveness (percent of errors detected during step) of the error detection process.

The above was true at a gross level. There were certain classes of errors that can only be found by certain activities such as hardware / software integration issues in the SAIL where software was tested with flight equivalent hardware. The table below illustrates the application of "Orr's laws" for two levels of average effectiveness.

Number of Sequential Steps	Total Percent Of Errors Detected For Single Step Effectiveness of 40 %	Total Percent of Errors Detected for Single Step Effectiveness of 60 %
1	40%	60%
2	64%	86%
3	78%	94%
4	87%	98%

Figure VII-6 Total Process Effectiveness by Number of Error Detection Processes

VIII. PASS Improvements as the System Evolved

With the computers in control of virtually all critical space shuttle functions, software changes were often considered as a solution to address risks and problems with the hardware sub-systems. Software changes also proved to be a cost-effective method of optimizing the behavior of each sub-system as operational data became available. When the cost and schedule associated with procurement and recertification of a hardware sub-system were considered, software changes often became a very attractive option.

Major Upgrades to Facilitate ISS Control

There were many examples of software being used to address shortcomings in hardware systems. One such example was the orbit flight control system (called the Orbit Digital Auto-pilot or Orbit DAP) where a series of software changes were implemented to help workaround the lack of sufficient low impulse Reaction Control System (RCS) jets to provide mated orbiter/station (ISS or Mir) control as the role of orbit flight control changed.

As designed and built, the space shuttle had 38 primary Reaction Control System (PRCS) jets and 6 vernier RCS (VRCS) jets. The PRCS jets, each having 870 pounds of thrust, were intended to provide coarse control while the VRCS jets, each having 24 pounds of thrust, were intended for fine control. Since the PRCS jets were also used for safety critical operations such as during separation from the external tank, rendezvous, proximity operations, docking, backup deorbit, and entry flight control, they were extremely redundant. The VRCS jets, however, were not anticipated to be as critical as they ultimately became and, thus, they were a zero fault tolerant system. The Orbit DAP was originally designed for an execution rate of 25 hertz. However, due to the combination of high system loading and the issue of hydraulic hammering in the RCS system, the Orbit DAP rate was reduced to 12.5 Hz and the minimum duration for firing an RCS jet was increased from 0.04 to 0.08 seconds.

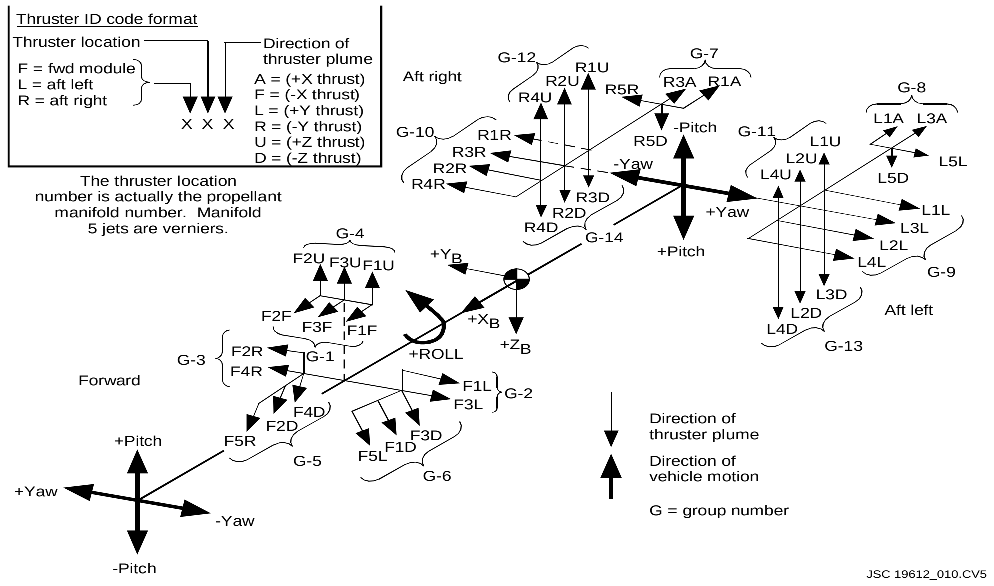

Thruster ID code format
Thruster location
F = fwd module
L = aft left
R = aft right
X X X
Direction of thruster plume
A = (+X thrust)
F = (-X thrust)
L = (+Y thrust)
R = (-Y thrust)
U = (+Z thrust)
D = (-Z thrust)
The thruster location number is actually the propellant manifold number. Manifold 5 jets are verniers.
Aft right
Aft left
Forward
G-1
G-2
G-3
G-4
G-5
G-6
G-7
G-8
G-9
G-10
G-11
G-12
G-13
G-14
R1U R2U R4U R5R R3A R1A R5D R1R R3R R2R R4R R3D R2D R4D
L1A L3A L5L L5D L1U L2U L4U L1L L3L L2L L4L L3D L2D L4D
F2U F3U F1U F2F F3F F1F F2R F4R F4D F2D F5R F1L F3L F3D F1D F5L
+Y_B
+X_B
+Z_B
+ROLL
-Pitch
+Pitch
-Yaw
+Yaw
Direction of thruster plume
Direction of vehicle motion
G = group number
JSC 19612_010.CV5

Figure VIII-1 Space Shuttle Reaction Control System

The primary Reaction Control System (PRCS) jets were designed to satisfy fail operational-fail safe translation and attitude control from main engine cutoff through on-orbit operations and entry. Initially, they were the only thrusters on the orbiter. The VRCS jets were added later to accommodate new requirements for attitude control during payload bay-mounted experiments (e.g., fine pointing and micro-gravity operations), for which jet redundancy was not deemed to be required. As the program progressed, the role of orbit flight control changed dramatically. With the advent of the Russian Mir and International Space Station Programs, the Orbit DAP was tasked with efficiently controlling massive, flexible structures that were attached to the shuttle while minimizing the structural loading associated with the attitude control firings. For both Mir and ISS, it was determined that the structural loading associated with the firing of PRCS jets using the DAP design at the time was intolerable in many cases. VRCS control was also challenging for the then current VRCS jet selection algorithm. Despite this challenge, through software improvements only to both the PRCS and VRCS DAPs, the Orbit DAP maintained the ability to utilize both the VRCS and the redundant PRCS jets (as backup to VRCS) for orbiter control while mated to Mir and for orbiter control while mated to the ISS structure throughout completion of the Space Shuttle Program. During the final shuttle flight, the Orbit DAP (VRCS or PRCS) provided the ability for the 105,000 kg orbiter to mate to the over 400,000 kg ISS and provide acceptable attitude control with acceptable ISS structural loads while providing the adaptability to accommodate a variety of visiting vehicles configurations.

In addition to a set of software tweaks that were implemented, the following major software changes were incorporated into the Orbit DAP in order to facilitate the mated control capabilities required by the Mir and ISS programs. By the end of the Space Shuttle Program, the software that made up the Orbit DAP was the largest and most complex of all the orbiter's Guidance, Navigation, and Control software.

1. *Alternate Primary Mode (ALT)* – ALT mode was added in 1989 to minimize the structural loading associated with PRCS jets as much as possible. In essence, ALT tried to make the redundant PRCS jets loads to the structure nearer to those produced by the VRCS by applying more computationally complex control algorithms and enforcing a set of constraints for PRCS jet firings. The ALT jet select logic evolved from the table look-up previously used for PRCS by implementing a dot product approach to the selection of PRCS jets, the approach previously used only for the VRCS jets. Loads reductions came through the implementation of firing constraints for the primary jets. These include:

- Maximum On Time – typically set to 0.08 seconds. This was the maximum allowable firing time before an enforced delay period.
- Minimum Delay Time – defined by structural analysis but typically set to over 5 seconds. This is the minimum period with no PRCS firings before another firing is allowed.
- Maximum Simultaneous Jet Firings – typically set to 1 for later missions. This was the number of PRCS jets that could be commanded on during each firing period.
- Jet Options –This allowed the selection of either tail-only, or both nose and tail jets for consideration by the jet selection algorithms. Additionally, the up-firing jets could be removed from consideration to prevent the pluming of the station.

Use of the ALT mode, while having higher loads than the VRCS, allowed for a back-up mode for ISS control in case of a failure of the non-redundant VRCS system and allowed the Orbit DAP to remain a viable control option through the completion of the ISS.

2. *Notch Filtering of Vehicle Attitude Estimate* – As the structures that the Orbit DAP was required to control became larger and more flexible, a notch filtering capability was added to the State Estimation software in 1992 to improve the efficiency of VRCS and ALT flight control performance. The ability to apply up to 6 notch filters to the state estimator allowed the software to filter out much of the flexible modes of the system and thereby minimize the oscillatory component in the attitude estimate. With a more accurate estimate of the steady state component of the orbiter attitude, the Orbit DAP could eliminate some unnecessary jet firings thereby reducing propellant consumption and reducing structural loading.

3. *On-Board Jet Acceleration Calculation* – The angular accelerations expected from the firing of each RCS jet with the current mass configuration was an important component to the Orbit DAP flight control algorithms. As originally conceived, these accelerations were calculated on the ground and provided to the flight control system as pre-calculated data. However, as the mass configurations and mass properties became less certain as the mated system became more complex, it became necessary to update mass properties and recalculate expected angular accelerations during the flight. In 1993, the capability for the on-board software to recalculate jet accelerations from uplinked and/or preloaded mass properties was implemented.

4. Minimum Angle Jet Select (MinAng) – MinAng jet select was added in 1994 to improve attitude control performance with mated vehicles. The original vernier jet select algorithm (Dot Product), used for the VRCS (and for ALT when the *Alternate Primary Mode* was added in 1989), selected the set of jets to fire that would provide the greatest angular acceleration in the desired direction with little consideration of the collateral effects on off-axes. The MinAng jet select algorithm, applicable to both VRCS and ALT, was a more computationally complex iterative algorithm designed to select jets that minimized off-axis accelerations.

5. *Automated ISS Reboost Capability* – An Automated Reboost capability was added to the Orbit DAP in 1998 to facilitate the use of excess orbiter propellant to increase the altitude of the ISS. With propellant a precious commodity on ISS, the ISS program had a strong need to get as much reboost as possible from the orbiter before it departed. Prior to this software change, the procedure was done manually and had some associated risks. It was also very crew intensive. The Automated Reboost capability allowed the Orbit DAP to command either VRCS or PRCS translational firings of specified durations, followed by specified intervals between firings, to slowly increase the

ISS orbital velocity while minimizing the structural loading imparted by the firings. This capability was designed to work in concert with the existing automated attitude control algorithms to command translational firings in quiescent periods where no attitude adjustments were required.

6. *RCS Auto-Manifold Close Capability* – As the ISS grew larger and more flexible, it was recognized that a single failed-on PRCS jet could cause a catastrophic rupture of the ISS before action could be taken to isolate it (the failed-on PRCS jet needed to be shutdown within 1.5 seconds). Although the likelihood of an uncommanded RCS firing was low and operational steps were taken to mitigate the remaining risk, a software change was required to allow the orbiter to continue to dock with ISS. A software change was implemented in 2005 to allow the flight software to automatically detect and confirm a failed-on RCS jet and immediately command closure of the manifold within the required 1.5 seconds to cut off propellant flow to the jet.

7. *Variable Delays for ALT Mode* – Although they were added in 2004, Variable Delays were first used in 2009. Starting with STS-127, projected structural loading due to PRCS attitude control firings drove an increase in the delay time that left the system without enough ALT authority to control the mated stack. A software change was enabled to vary the duration of the RCS firing delay to spread the energy associated with the ALT firings over multiple structural modes thereby reducing the loading on any single interface (e.g., rather than a delay always producing a 10 second delay it would produce delays of 10 ,12, 8, 14, 6, 7, 13 seconds that would average 10 seconds). This allowed the average ALT delay time to remain low enough to keep the ALT as a viable control option for the remaining flights.

8. *Notch Filter Feed Forward* – Beginning with STS-120/10A in 2007, the ISS began a rapid build out of its truss. This increased the flexible dynamics and affected attitude control. The increased flexibility required increased attenution from the notch filters of the Orbit DAP State Estimator. Simply increasing attenuation in the notch filter design to prevent this dynamic interaction was not practical, for the price of notches was increased lag in the State Estimator. The size of the required notches would have increased propellant usage dramatically. By feeding forward the expected rate changes from the jets directly into the notch filter software, this lag penalty was virtually eliminated and rigid body performance was restored, thus enabling Orbiter control for STS-120 and later flights.

9. *Flight Control Modifications to Reduce VRCS Shelf-Pulsing* – During the post-docking rotational maneuver on STS-114/LF1, two VRCS jets each fired nearly 500 times in order to accelerate and decelerate the mated stack. As ISS was to grow dramatically in size, this pulsing behavior threatened to violate a duty cycle constraint that limited the number of pulses each VRCS jet could fire in an hour. In response, a FSW change was implemented to allow the control logic to detect this rapid pulsing and adjust the flight control algorithm to command fewer, longer pulses in order to protect the hardware. As a side benefit, the algorithm was used to de-tune the pulsing from the ISS major structural modes.

Major Upgrades to Improve Abort Outcomes

Throughout the Space Shuttle Program, a major focus was on improving the survivability of abort scenarios since main engine failure was assessed as a major program risk. For STS-1, two abort modes were designed and certified:

- A Return to Launch Site (RTLS) Abort could be performed early in Ascent and executed a series of complex procedures to pitch the vehicle around, arrest it’s downrange velocity, safely separate from the External tank (ET) and, then, safely land at Kennedy Space Center (KSC)
- An Abort to Orbit (ATO) could be performed late in Ascent to retarget the guidance software to a minimal orbital trajectory rather than the desired operational orbital trajectory.

The RTLS procedures were challenging and would bring the mated Orbiter/ET system close to structural limits as it reversed direction to return to KSC. Through the life of the program, only one engine failure was encountered (actually a sensor failure – the engine was working fine) which resulted in an Abort to Orbit. Other than that Abort to Orbit, the bulk of the abort software was never executed in flight. However, much of the calculated reliability of the Space Shuttle System was predicated on the ability to successfully execute aborts to address failures encountered during Ascent.

During the first four orbital flights, most contingency (two and three engine failure) scenarios would result in loss of vehicle and ejection seats were included to mitigate the risk to the two person crews. But, starting with STS-5, the ejection seats were disabled as they could not protect a crew of four and other Contingency Abort concepts were developed and refined. These Contingency Aborts introduced challenging new FSW requirements and crew procedures that were designed to get an impaired orbiter to a safe landing or at least a survivable bail-out. During the stand-down period after the Challenger accident, a bail-out concept was designed where an orbiter that could not achieve a run-way landing would be flown straight and level over the ocean at a slow enough speed that the crew could egress the shuttle and parachute to safety. As the program progressed, Contingency Abort improvement and automation became a focus in order to improve survivability for these complex, time critical procedures and to reduce the amount of training time necessary to devote to them considering the relatively low likelihood of Contingency Abort.

Virtually all of the abort improvements involved either refinement of the manual procedures the crew performed or improvement of the flight software that either automated those procedures or improved the GNC performance. And, as can be seen from the "Abort Survivability for High Inclination Flights (Notional)" chart below, the results were astonishing. By the end of the Program, all black zones had been eliminated for two engine out scenarios, and the majority of three engine out black zones had been converted to bail-out scenarios.

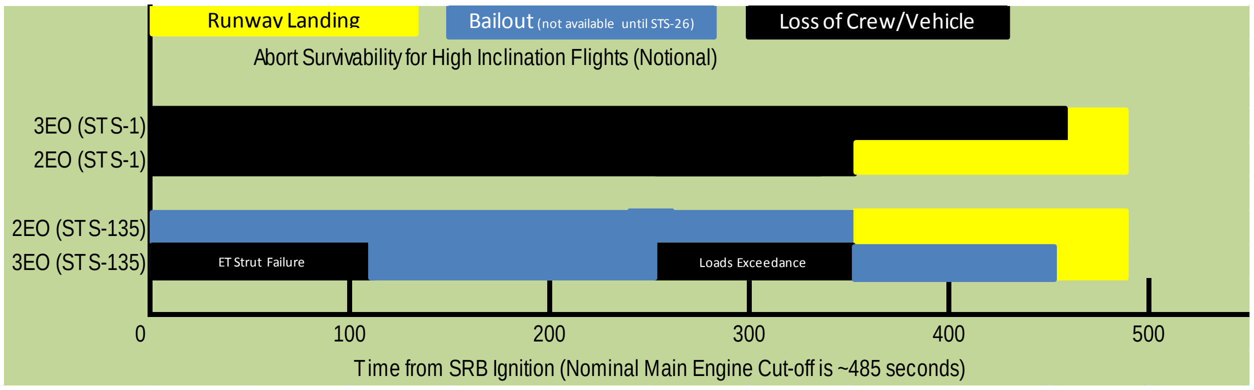


Figure VIII-2 Abort Survivability for High Inclination Flights (Notional)

The Evolution of PASS Abort Software

After the basic Ascent and Entry capabilities were proven during STS-1 and the resultant performance data was closely analyzed, improvements and expansion in the abort software began with STS-2. As flights continued, more data became available and more abort concepts were refined. Although there were literally hundreds of changes put into the FSW that were intended to improve abort performance, the following list is a short summary of the major software changes that were implemented into the PASS FSW in order to bring about the safety/survivability improvements.

- Trans Atlantic Abort (TAL) Capability – After STS-1, a TAL capability was added that provided the guidance and control necessary to facilitate a European or African landing if engine failures occurred too late in the ascent profile to make RTLS an effective option. Addition of TAL as a certified abort mode drastically closed the black-zone (region of unsurvivability) for the period where the orbiter had too much energy to return to Florida (RTLS) but did not have enough energy to achieve a stable orbit (ATO). By the end of the Program, the TAL became the preferred abort mode since it eliminated the complexity and stresses associated with reversing direction. The TAL acronym was later changed to Transoceanic Abort Landing when Vandenberg launches were envisioned.

- Main Propulsion System (MPS) Dump Capability - After STS-7, software changes were implemented to automatically command RCS/OMS propellant dumps through the main engines to improve vehicle approach and landing performance by moving the Center of Gravity (CG) to a more stable/controllable location.

- Automation of Normal Acceleration (Nz) Hold Maneuver – After STS-28, a software change was added to improve guidance and automate one of the most difficult contingency abort maneuvers. The Nz Hold maneuver requires the orbiter to fly very close to the maximum g forces it can structurally withstand and was very difficult to fly manually under the best of conditions.

- Single Engine (2EO) Contingency Abort Procedure Automation – In 1992, a major software change was added to PASS to automate the complex procedures required to successfully perform the powered flight portion of 2EO contingency aborts while providing appropriate display and telemetry information to allow the crew and ground to monitor the software and it performed the required maneuvers. This change also introduced the Autoloft capability which changes trajectory to maximize altitude upon detection of two engine failures in order to reduce dynamic pressure at ET separation. These changes significantly improved the survivability of 2EO contingency scenarios while also reducing the training requirement and, thereby, allowing training time to be refocused on more likely scenarios.

- Abort Sequencing Redesign – In 1992, an area of the FSW that had a large set of discrepancies written against it was redesigned with the goal of meeting the existing requirements in a much more robust and maintainable way. This was a major re-coding effort that took advantage of process improvements to produce a more centralized and organized set of software that proved to have a much better quality record than its predecessor.

- TAL Droop Capability – In 1993, TAL Guidance was improved to actively manage the minimum or "droop" altitude that can occur with the loss of an engine during a TAL. If the Altitude drops too low before the orbiter obtains sufficient velocity to resume climbing, it can violate heating constraints and rupture the External Tank. The guidance algorithm controls pitch angle, throttle and Main Engine Cut-Off (MECO) targets to insure that the altitude remains above 265,000 feet, if possible, and alerts the crew if it is not.

- Low Energy TAL Automation – Another TAL Automation change was implemented in 1993 that created a new low energy algorithm that commanded a set of automated maneuvers that could stretch the trajectory enough to convert a set of scenarios that were previously bail-out cases to reach a trans-Atlantic runway.

- Three Engine Out (3EO) Automation – In 1997, the FSW was changed to automate and improve the decision and maneuver logic associated with 3EO scenarios. This change advanced 3EO control from a manual cue card based set of generic maneuvers to an automated set of maneuvers customized to the particular state of the orbiter. Although 3EO scenarios are often unsurvivable, this logic optimized ET separation conditions, Center of Gravity (CG), and attitude control to give the crew the best chance at survival while significantly reducing costly training requirements.

- East Coast Abort Landing (ECAL) Automation – In 2001, the final major Abort Automation change was completed with the automation of East Coast Abort Landings. As the Shuttle manifest became dominated with flights to Space Station, the vast majority of flights were high inclination flights that made East Coast aborts a much more likely and relevant abort mode. Like the other abort improvements, this automated abort procedures to maintain the orbiter within physical limits while accurately performing optimal maneuvers to maximize the likelihood of making an East Coast runway with survivable conditions.

- Ascent/Entry Bearing Displays – In 2007, a major situational awareness improvement that took advantage of the MEDS "glass cockpit" display capabilities was implemented. This change introduced two complex moving-map displays that showed the Orbiter's position and trajectory states along with some predictive information. The Ascent version showed East Coast and Trans-Atlantic Landing abort sites and was useful in selecting appropriate aborts in the event that communications was lost with the ground. The Entry version showed the vehicle's energy state and provided navigation data that was useful in monitoring vehicle energy state and performance as it approached the selected abort site.

- RTLS ET Sep Improvements – In 2007, the last major abort improvement was implemented to address the most probable set of loss of control and External tank recontact scenarios. Changes were made to Guidance, Flight Control, and Sequencing to add an improved pre-separation pitch down maneuver to propel the ET away from the orbiter at separation and a set of more aggressive post-separation flight control maneuvers to increase the likelihood of maintaining control after the challenging ET Separation maneuver.

Major Upgrades to Improve Crew Situational Awareness

Efforts have been made throughout the Space Shuttle Program lifetime to improve the information available to the astronaut crew. Much of this information was unique to Shuttle since the same human-rated vehicle was used as a launcher, on-orbit platform and winged re-entry vehicle. The crew members were integral parts of each phases of every Shuttle mission, although many of these same functions could be performed automatically.

During the Space Shuttle's design phase, a number of options were reviewed which would have included a large number of CRTs for a glass cockpit-like display suite. The limited computing power of the 1970s prevented those prototype cockpits from becoming reality.

Beginning with the orbiter Enterprise (OV-101), a standard three-CRT forward cockpit with supplemental electromechanical instrumentation was used. A single aft CRT with a subset of electromechanical instrumentation was also included. The CRT system, which was formally known as Multifunction CRT Display System (MCDS), used a Display Electronics Unit (DEU) to format and generate graphics. These graphics were then shown on a Display Unit (DU). The DU was vector driven, which meant that the output of the DEU was the equivalent of voltages used to drive the CRT guns. Since the CRT was an analog-driven system, it lent itself to being able to make major graphical movements, such as rotating the entire display format, with simple commands.

However, this system was also limited in speed. Each movement of the electron guns required a set amount of time and if new commands were given prior to the completion of the gun movement, a jumbled display could result. The display graphics were also monochrome (green and black), which was not unusual at the time of the Shuttle debut but became a more limiting factor as time progressed.

The DEU included a microcoded instruction set that converted simple 16-bit instructions into analog movements of the DU electron gun. The DEU interface to the GPC was through a Multiplexer Demultiplexer attached to the MIA bus. The DEU also had an interface to the Keyboard Unit (KU) which allowed the astronaut crew to interface with the GPCs.

The DEU included random access memory (RAM) that stored the instructions to be performed. These instructions were loaded by the GPC – in effect the DEU would be programmed in real-time by the GPC. Changes to a displayed number, for instance, would require rewriting the specific portion of DEU memory that stored the instructions to generate the number to be displayed. The DEU would then perform a scan through memory, read the stored instructions, move the electron gun accordingly and update the displayed number. This was done twice a second for PASS displays and three times a second for BFS displays.

The electromechanical instrumentation consisted of stenciled metallic tape the could be spooled to show the correct values at a lubber line. The spooling was done by providing a voltage (usually in the range of 0 Volts to +5 Volts) to spool the tape based on General Purpose Computer (GPC) output. The spooling was accompanied by the sounds of the tapes moving to the correct position and this provided a unique sound in the Shuttle cockpit as the electromechanical tapes were initialized.

The Attitude Director Indicator (ADI), which showed the current attitude of the orbiter, and the Horizontal Situation Indicator (HSI), which acted as an advanced compass and runway locator, both were driven by voltage signals originating with the General Purpose Computers (GPCs). For example the ADI would roll to a specific location based on a roll voltage applied to the ADI ball. For the roll position the ADI would receive a voltage that corresponded to the sine function of the current roll angle as well as the cosine function of the current roll angle. Pitch and Yaw functioned in the same way. This was the same system used to drive the ADI ball in the Apollo command module.

One of the chief limitations of this display suite was that the crew members had to look below the windows to see the information. This made the task of crew members "manually" flying the final four minutes before landing more difficult. This problem was solved with the first display addition to the Shuttle orbiters – the Head-Up Display

(HUD). The HUD was added to the fleet with the delivery of Challenger (OV-099) in 1983 and retrofitted onto Columbia (OV-102) a year later. All other orbiters came with the HUDs already installed. The HUDs were driven by PASS data provided by the GPCs. An update to BFS in 2006 allowed the backup software to also drive the HUDs.

An identical HUD was provided to the Commander and Pilot station. The HUD showed the current pitch and roll as well as an outline of the runway (when in sight), equivalent air speed and current altitude. Piloting cues were also provided, such as an indication of when to flare the vehicle in pitch prior to landing. One other key item provided was an indication of the attitude GPC-computed guidance versus the currently flown attitude. The interface to the GPCs was done by use of two HUD-unique data packets sent by the GPCs to the HUDs over the flight critical MDM Interface Adapter (MIA) databus. The HUDs would also "listen" to the data packet messages being sent to the electromechanical ADIs to determine the current vehicle attitude.

The HUDs were purchased by NASA from Kaiser Electronics, of San Jose, California, as a "black-box" with completed hardware and software. The HUD was not expected to be changed or updated after delivery, although the capability to do field updates existed. However, this capability was not used after the updated HUD display formats were flown on STS-8.

It was with this cockpit layout that in the early-1990s, the Space Shuttle Program began to plan one of its largest upgrades – the glass cockpit. The glass cockpit was formally known as the Multifunction Electronic Display Subsystem (MEDS). Very detailed renderings of the MEDS cockpit layout were provided to potential vendors in the 1992-timeframe. These renderings showed a cockpit layout consisting of nine forward flat-panel displays with a square aspect ratio.

In addition to these forward displays, the MEDS upgrade consisted of two aft displays. These displays replaced all of the CRTs and electromechanical instrumentation in the cockpit. Originally the MEDS upgrade was to be performed in two phases – one that replace the CRTs and then a second that would replace the remaining electromechanical instruments. For schedule and easy-of-installation reasons, this plan was replaced with one that replaced all of the CRTs and electromechanical instruments at the same time.

The MEDS hardware consisted of nine forward LCD displays and two aft LCDs using state-of-the-art mid-1990s display panels. These were known as Multifunction Display Units (MDUs). The MDUs were driven by four Integrated Display Processors (IDPs). The IDPs performed a similar function as the original DEUs and the MDUs performed a similar function as the DUs. The difference being that the IDPs also received all of the data necessary to draw graphical representations of the original electromechanical instrumentation as well as mimicking the function of the DEUs to draw GPC-generated display formats. Some of this data was provided by the IDPs listening on the flight critical data busses and other pieces of instrumentation data were gathered using analog-to-digital converters.

While the IDP was connected to the Shuttle databus using the standard Multiplexer Demultiplexer (MDM) interface connected to the MIA bus, the communications within the MEDS system was performed with a MIL-STD-1553 interface. This allowed for an industry-standard interface to be used within MEDS. Similarly the software in the IDP and MDU were both programmed in industry standard languages. Ada was used throughout the project with C language imported, as needed.

During the MEDS development timeframe, the reliability of the original cockpit displays started to fall. For instance, during the STS-75 launch in February 1996, an electromechanical gauge showed that one SSME remained at 40 percent of rated thrust during ascent while telemetry showed the SSME was working normally at 104 percent of rated thrust. The supply of spare electromechanical instruments and CRTs were also become a concern as more spares were used.

The first flight of MEDS took place on mission STS-101 in May 2000 aboard the orbiter Atlantis (OV-104). The MEDS hardware and software performed without any issues during the flight. Later Discovery (OV-103) and Columbia (OV-102) were upgraded to MEDS. It was only after the STS-107 Columbia accident that Endeavour (OV-105) finally was upgraded to the MEDS system, with Endeavour's first MEDS flight occurring on mission STS-118, in August 2007. At that point, the entire Shuttle orbiter fleet had been converted to MEDS.

To minimize training with a mixed-fleet of MCDS and MEDS orbiters, one of the ground rules of the MEDS development was that it had to mimic the original MCDS cockpit, including having the MEDS graphics replicate the colors used on the original instruments. While this provided a short-term cost savings for training, it stretched-out the time needed to take advantage of the new MEDS capabilities.

The limitation on using the new MEDS capabilities also meant the GPC-to-IDP interface retained much of the same instructions as the DEU implementation. DEU instructions that were found not to have been used by PASS or BFS in any display when MEDS began were eliminated. The other instructions were modified to support the raster scanned MDU LCDs. The conversion from vector-based graphics to raster-scanned graphics was difficult at times, however to the PASS and BFS software the change from MCDS to MEDS was transparent.

The PASS and BFS flight software included a single bit that could be set to indicate that the orbiter was upgraded to MEDS or still had MCDS. This would have allowed for upgraded MEDS capabilities to be used even in a mixed-fleet environment. However, it was not until OI-30 in the mid-2000s that true MEDS-only capabilities were introduced with flight phase-unique enhancements to the ADI and HSI graphics, called the Ascent / Entry Primary Flight Display (AEPFD)

This introduction of additional MEDS capabilities occurred after the cancellation of a follow-on cockpit display upgrade called Cockpit Avionics Upgrade (CAU). CAU would have replaced the four MEDS IDPs with three Command and Data Processors (CDPs) along with new CDP-based display software. The MEDS MDUs would not have been replaced. The PASS and BFS software would have been extensively modified to support the CAU interface. Due to the decision to end the Space Shuttle Program (at that time) in 2010, CAU was cancelled.

However, much of the completed CAU software work was reviewed for use in MEDS. While the requirements and software could not be used as-is because of the different architecture and hardware, many of the remaining MEDS upgrades were derived from this CAU work. The Bearing Displays were an example of this re-use. In order to implement these new displays, new instructions were added to the GPC-to-IDP instruction set, including color, new characters and overlay display backgrounds that could be called up with a single command. Details of the Bearing Displays are provided in the previous section of this paper.

One of the CAU project's goals was to reduce the reliance on the GPCs for display processing. This would have freed-up memory in the GPC for planned GN&C additions and further delay any need to replace the GPCs. Similarly once CAU was cancelled, MEDS looked at ways to reduce the memory footprint for display processing in the GPCs. The first step of this reduction moved much of the background display processing for two GPC-generated displays completely into the IDP. These displays were the PASS Horizontal Situation display and the PASS DPS Utility display. The move from GPC-processing to IDP-processing saved approximately 4 percent of the PASS GPC's ascent memory. Further use of this memory-saving (or more accurately memory swapping) technique could have returned over 10 percent of GPC memory for use in improving Shuttle GNC capabilities, such as abort options.

A number of planned MEDS upgrades were not implemented because the program ended before they could be implemented. One upgrade that almost made it onboard was the MEDS Vehicle Status Display (MVSD). MVSD would have combined PASS and BFS information together on a single MEDS display and shown an overall summary of the vehicle's current condition. While MVSD was one of the most worthwhile planned upgrades, its planned completion date was simply too late in the overall lifecycle of the Space Shuttle Program. While the software to provide the MVSD capability was completed and fully tested in time, there was not enough time to add MVSD to the Space Shuttle training flows and therefore the MVSD display never flew.

The only other planned MEDS "big-ticket item" would have moved all processing for GPC-displays (PASS and BFS) into the MEDS IDP. The GPC would have simply "told" the IDP which display to bring-up and the MEDS system would have handled all of the processing for the display, with the exception of keyboard processing. This design, called Virtual Data (VData) would have provided maximum GPC memory savings while also allowing all of the capabilities of the MEDS system to be used on the formerly-GPC hosted displays. Alas, the time remaining for Shuttle flights did not permit implementation of the VData concept, however VData was extensively prototyped.

The MEDS upgrade turned out to be one of the largest upgrades in Space Shuttle Program history. The MEDS upgrade extensively gutted the forward portion of the cockpit. But once installed, the MEDS upgrade retained backwards compatibility for all interface systems and allowed for a near-seamless software integration and was truly a successful upgrade to the Space Shuttle's capabilities.

IX. Lessons Learned

This section provides a summary of the most significant lessons learned from the Space Shuttle flight software team over the life of the Program. Also included at the end of this section are the lessons learned from the investigation of two specific software problems that made it through the development and test processes into the operational system.

a) Strong Processes
As documented in earlier sections, one of the major contributors to the success of the shuttle flight software was the implementation and adherence to well-defined, documented, disciplined and controlled processes with a focus on quality and improvement. Future programs should perform extensive analysis of errors to identify root cause, determine corrective actions and make process improvements. There should be independence between development and verification personnel and verification personnel should participate throughout the lifecycle (requirements definition, design, code inspections, testing, etc.). Comprehensive testing should include nominal and off-nominal scenarios utilizing flight-like hardware and full software loads in high-fidelity test facilities under formal configuration control.

b) Operational Scenarios
All possible operational scenarios, both nominal and off-nominal, which the software could be exposed to should be identified, addressed in requirements, accommodated via design/code, and tested. It is important to insure that proper initialization will occur under all scenarios supported by the software. Requirements should address what is to be done for failed hardware under all scenarios supported by the software. Scenario analysis should identify the maximum ranges for parameters under all scenarios. Variable precision should correctly support the maximum ranges. Many scenarios-related problems have extremely small timing windows. This is very unlikely to be detected during testing only and may require "Multi-Pass" analysis methods to insure identification. Robust scenario testing should be implemented. Adequate test facility resources are required including resources for off-nominal testing. It is important to maintain communications with operations personnel to insure that, when testing, the scenarios considered model the expected operational use of systems capabilities as closely as is possible.

c) Interface Testing
Software Interface Control Document requirements should be explicitly verified in an end-to-end manner. Two PASS in-flight software problems were due to the failure to completely verify the PASS Systems Management (SM) to Spacelab ICD. Both required in-flight patches due to mission objective impacts for specific payloads.

d) Know the Limitations of Your Target Processing Environment
As the AP-101 architecture aged, the processing power that many engineers had on their desks surpassed that of the on-board system. There were instances when issues such as processing speed and arithmetic precision delayed schedules and even impacted flights. It is important to remember the performance limitations of the digital system and, in some cases, to model them when designing new software and algorithms.

e) Analyze Error Logs
Anomalies occurring which result in signatures being written to software error logs may go undetected unless the software error logs are analyzed after the run. Procedures should be documented to always require analysis of software error logs after completion of the test. The test may be designed to automatically stop if certain types of error conditions occur. This allows for capturing full software memory at the time of the error condition.

f) Hire Talented People and Train them Well

Although the PASS software processes were widely acclaimed, the results were only as good as the people who executed them. Over the years, there were many extremely talented people who designed, developed and maintained the PASS system. The challenging work, cooperative environment, and enjoyable working conditions encouraged those people to stay with the PASS project. As those experts passed on their knowledge, they established a culture of quality and cooperation that persisted throughout the program.

g) Simulation Models
It is important to collect appropriate data during simulated hardware tests so that any anomaly occurring is identified. Models in the software test environment must provide valid outputs. Timing related software problems will be impossible to detect in the simulation unless the hardware models provide random variation similar to actual hardware characteristics. Using checkpoints and restart capabilities may additionally limit the number of opportunities to observe timing related software problems even if the simulation supports random hardware timing variation. However, the use of checkpoints and restart capabilities is highly desirable. These capabilities allow for efficient use of test facility resources. These capabilities also greatly enhance the ability to duplicate problems by providing a duplicate software/environment state just prior to the event being duplicated.

h) Hardware / Software Integration Testing
It is extremely valuable to utilize flight or flight-like hardware in end-to-end hardware/software integration testing. The shuttle flight software team followed the principle of "test like you fly" by utilizing flight-like GPCs and other hardware in integrated testing. Getting as close to actual hardware performance and timing is essential. Using checkpoints and restart capabilities may limit the number of opportunities to observe timing related software problems. Multiple tests from the same source checkpoint will have one fixed set of software internal timing relationships. Appropriate data should be collected during hardware tests so that any anomaly occurring is identified. Latent defects can remain in the software for years until the scenario and hardware re-action timing align.

i) Manual Processes
Many processes such as late updates to flight reconfiguration are initially done manually. Three in-flight software problems in the 1983-1985 period were introduced this way. Manual processes require continuous management oversight to insure rigorous and correct execution.

j) "Apparently Unrelated" Changes
Multiple "apparently unrelated" changes can collectively produce unexpected erroneous consequences. Regression testing is required to insure software functions continue to work correctly. There is an ever present risk to "stumble" into maintenance traps once the maintenance trap is introduced into the software. A maintenance trap is typically the result of waiving programming standards for some small short term savings in implementation time, code space, or computer performance. These short term savings are insignificant compared to the long term project cost incurred during the maintenance phase.

k) Formalize Management
It is essential to formalize management and lead analysts' responsibility for assessing skills proficiency and work performance history for every individual on every team and evaluate risks based on the skills mix with closed loop responsibility to the program manager. Every change to human rated flight software must be implemented with the same professional attention to detail by knowledgeable and motivated personnel.

l) Collect Measurements
It is essential to collect measurements data and proactively analyze data to search for "in process" symptoms such as pre-build errors found in inspections by only one inspector.

Contributor To PASS FSW High Quality	Context
Multiple releases and multiple iterations of testing prior to STS-1.	Delays in launch date due to TPS and SSME issues provided more testing time and more opportunities to fix identified problems.
Fully automated Flight-to-Flight Reconfiguration Process and Tools	Early flights had a number of System Management in-flight failures due to late manual updates.
Structured "PASS Revalidation" activities between Challenger accident and STS-26	Direct contributor to eliminating Severity 1 (Loss of crew/vehicle) DRs from PASS
Continual enhancements of the Requirements/Design/Code/Test Inspection Processes	▪ Have appropriate participation in each type of inspection including external community participation ▪ Having appropriate re-inspection criteria
Adequate test facility functionality and capacity (equipment to execute cases on flight equivalent hardware)	Significant improvement in in-flight reliability between STS-51L and STS-26 during a period when test facility capacity increased by a factor of 3.
Defined criteria for selection of personnel for teams; define how to resist over commitment of critical skills.	Critical skills management has always been a priority. Re-enforced by action From OI-25 PTI DRs where team skill and over commitment were contributing factors.
Rigorous configuration management and control of all products including requirements, design, code, and tests.	Basic necessary condition

Figure IX-1 Contributors to PASS Flight Software High Quality

What Can Be Learned from PASS' Most Critical Error?

Risk is an inherit component of developing software for a manned space vehicle like the Space Shuttle. Every available action was taken to eliminate risk. Every issue identified was pounded flat before a system was certified as ready for flight. Historically, the moment of greatest risk to a crew from a PASS fault occurred on June 26, 1984 when the STS-41D launch countdown was aborted at T – 6 seconds when PASS detected an anomaly in orbiter's main engine number three. Without the fortuitous SSME anomaly, STS-41D would have launched with about a 1 in 6 chance of being unable to separate the Solid Rocket Booster (SRB) or the External Tank (ET) which could have resulted in the loss of crew and vehicle.

Background

Each GPC had a separate CPU and Input Output Processor (IOP) which allowed inputs and outputs to occur in parallel while software processing was on-going. There was a set of critical output commands (e.g., ET separation commanding) that had to be calculated and output during the same 40 millisecond software execution in order to meet stringent timing requirements. Because of this, the IOP processing was skewed to begin issuing critical outputs 10.4 milliseconds after the critical software that calculates the commands began running. The software was designed to insure that all calculations of critical outputs were complete before the outputs begin. However, in a multi-tasking, interrupt driven system, it could be difficult to insure that timing constraints such as these were effectively enforced.

How Did the Error Happen?

At the time of this error, computer CPU resources were very limited. As changes were made, it became very important to identify where and how to make those changes to minimize impacts to the system performance and to insure that the resultant system performance was acceptable. To insure system constraints were enforced, performance testing was executed on every release. But, since it was not possible to test all possible timing scenarios, a representative set of scenarios were run. Output margins were checked on the system where the error was introduced (OI-3, released on 09/01/1983) and observed to meet required margins. Due to manifest changes, no flights were flown using OI-3. The next release (OI04, released on CI 12/20/1983) was again verified with measurement of output commands with adequate margins observed. STS-41D was the first flight scheduled to use either OI-3 or OI-4. The launch was attempted on June 25 and June 26 with scrubs and a pad abort due to hardware failures. Discovery returned to OPF and number three main engine replaced. STS-41D was rescheduled for late August. In parallel, verification testing had proceeded on the next release (OI-5 with a release date of 06/08/1983). As data analysis continued, it was observed that the margin for critical outputs was inadequate in one tested scenario. This could result in a critical command being issued later than intended.

With a potential problem identified, it was documented with a Discrepancy Report (DR). Analysis began to determine the worst case results of this condition. It arrived at the conclusion that there were a set of module phasing conditions that could cause a critical ET separation command to be output to the hardware before it was updated resulting in a 40 millisecond delay in command issuance. Originally, it was believed that separation would occur if the output was delayed for 40 milliseconds based on expected hardware behavior. To insure this, a request was made through the NASA Mission Evaluation Room (MER) process:

Subject: MEC (Master Events Controller) Operation for SRB and ET Separation

Text: Ongoing Flight Software analysis indicates that there is a possibility that at SRB Separation and/or ET Separation the FIRE 3 command to the MEC may be issued 40 milliseconds after the FIRE 1 and FIRE 2 commands are first issued rather than on the same 40 millisecond output cycle. When the FIRE 3 Command is issued, the FIRE 1 and FIRE 2 commands will still be issued cyclically. It is our understanding per informal discussions with the technical community that MEC Operations during these separations will not be adversely affected. Please concur.

This request led to tests being performed on actual MEC (Main Engine Controller) hardware. The results surprised everyone. Test showed that SRB and ET separation would not occur for this scenario.

The NASA community was notified of the potential for an insufficient timing delay between PASS FSW-computed SRB separation command and the actual Timer-initiated output of that command to the Master Events Controller hardware which actually detonates the pyrotechnics. Analysis showed that it was theoretically possible for the actual hardware output to occur prior to command computation, thereby missing the separation command. Additional community analysis revealed that the BFS would be unable to separate the SRBs if the PASS condition occurred. System requirements had not adequately considered this scenario relative to BFS. System requirements resulted in only a four second window where the BFS could be engaged and complete separation. The window closed when PASS disconnected 28 volt power to the SRBs thereby eliminating the BFS ability to separate the SRBs.

Because of these issues and another that was being worked, the third launch attempt scheduled for Aug. 29 was delayed while work continued on this problem. Software engineers designed, developed, and verified a software change to assure all three booster fire commands were issued in the proper time interval.

The following timeline shows multiple critical changes made to PASS prior to the launch of STS-41D:

- During Aug. 18-23, 1984, NASA identified hazardous condition for abort landing case due to excess residual liquid hydrogen in orbiter aft plumbing. FSW assistance requested in designing a software solution. Several solutions provided for NASA review and selection of one. FSW designed, tested and delivered a 25 half-word code patch for the dump sequencer to evacuate the LH2 through an 8 inch fill and drain valve automatically once the vehicle slowed to 4500 ft/second
- During Aug. 27-29, 1984, IBM identified an insufficient minimum timing delay between PASS FSW-computed SRB separation command and the actual Timer-initiated output of that command to the Master Events Controller hardware which actually detonates the pyrotechnics. Analysis showed that it was theoretically possible for the actual hardware output to occur prior to command computation, thereby missing the separation command (also true for ET separation). Significant FSW analysis effort was expended in analyzing module-by-module timing data to identify potential solutions. Ultimate solution for STS-41D was to force two high CPU processes to execute on mutually exclusive computation cycles. FSW

designed, tested, and delivered a 12 half-word code patch to ensure a sufficient timing delay between the PASS-computed commands and the output of those commands to the hardware.

- During Aug. 29-30, 1984, NASA discovered a four-second limit on the BFS engage window following PASS-attempted SRB separation due to a PASS requirement to disconnect 28 volt power to the SRBs. FSW originated, designed, tested, and delivered a half-word code patch to eliminate the disconnect command for the SRB power. This resulted in an unlimited BFS engage window as intended.

With each of these software changes on-board, STS-41D successfully launched on Aug. 30 after a six minutes, 50 second delay when private aircraft intruded into warning area off coast of Cape Canaveral.

Lessons Learned

Although this particular situation turned out well, this issue represented the biggest threat to crew and vehicle ever, to our knowledge, caused by the PASS FSW. Although a fortunate slip kept this issue from flying, the mature process and the determined analysts indentified this complex issue and were able to remedy it. The safety culture of NASA software development prevailed in this case. The software analysts kept a system perspective and were willing to go outside the software world to understand how the software behavior was affecting the Space Shuttle systems. And, although the initial response from the hardware engineers was that the 40 millisecond delay would not affect the pyrotechnic hardware, they followed it up with a hardware test that showed this was not always the case.

What Can Be Learned from PASS' most Recent In-Flight Error?

When considering the effects of the world-class CMMI Level 5 rated flight software process utilized for developing and sustaining the PASS, and the exceptionally low PASS software error rate relative to the software industry, there were very few opportunities available for reviewing an error that made it through the process. This also reduced the opportunities to analyze how the problem was introduced and how it avoided detection. It was very interesting to investigate those few problems that actually slipped through this robust process and then glean the lessons learned that were discovered there. In doing so, we found we must always strive for discipline and thoroughness, because problems can somehow always surface, even in the best of processes, executed by the most talented and exceptional individuals.

One such software problem was discovered in-flight during STS-126 in November of 2008. Shortly after STS-126 achieved a stable orbit, ground controllers observed that two Systems Management capabilities that relied upon the Ground Interface Command Logic Controller (GCIL) were not functioning; the capability for automatic transition between two communications modes (S-band and Ku-band) and the capability for automatic transition of the Payload Signal Processor (PSP) to a new configuration (port change) when the Orbiter docked to the ISS. Each of these problems was merely a nuisance to operations and, fortunately, the impacts of these problems could be overcome by operational work-around procedures. This temporarily impacted communications with the Multipurpose Logistics Module (MPLM) and delayed completion of the full MPLM environment check by one day. It also required ground controllers to manually command handovers between S-band and Ku-band. However, as was always the case when a flight software question or issue was raised, the expert flight software team investigated, identified and explained the problem and its ramifications with complete thoroughness and in an impressively short amount of time, therefore mitigating the impacts. Thus, the first good lesson learned from this story was to make sure you always had the best team possible on hand to support critical operations.

A number of circumstances had to line up just right for this problem to slip through the many inspection, development, and verification opportunities to identify it. In fact, the subtleness of the problem was one contributing factor in the error being missed. Nonetheless, a series of conditions were required for this rare error to occur. The first significant factor was that this error was in the Systems Management (SM) Software. The SM software load was, by definition, less critical that the GNC software loads. The SM software load was designed to run only in a single GPC as opposed to GNC's ability to execute multiple simultaneous copies in a redundant set. In keeping with the Fail Ops/Fail Safe philosophy, no life critical functions could be included in the SM load since a single GPC failure would, at least temporarily, disable all SM capabilities.   Because of this reduced criticality, the robustness of simulations and hardware models for the SM software was often less than that of GNC components. This was a conscious decision to focus resources on the most critical applications.

The problem was tagged as “MER-08”, the eighth problem encountered during the STS-126 mission to be investigated by the Mission Evaluation Room (MER) team of civil servant and contractor engineers. The problem was later identified as software Discrepancy Report (DR) 126366, entitled: System Management (SM) General Purpose Computer (GPC) Failure To Send Ground Interface Command Logic Controller (GCIL) Commands.

As you would expect, the first thing that the MER managers wanted to check was whether there could have been other more critical commands corrupted in the same way either by the identified software change or any other previously implemented changes. The flight software team performed audits to ensure that this was indeed an isolated case. Once the operational work-around procedures were ready, and the audit was completed to ensure there were no other similar software problems, the attention of the flight software team was focused on the root cause investigation for the error and understanding how it could have been missed in development and testing activities.

Having identified the root cause of the problem, the flight software team then performed an oversight analysis to determine how this process escape occurred (i.e., how was it overlooked in development and test activities). This included review of all inspections and testing that was performed for this change, and analysis of the fidelity of PSP, OIU and GCIL hardware models/simulations that were utilized in testing. The results identified a number of contributing factors.

Development/Design and Code Inspection Process -The problem was traced back to an error introduced by the implementation of a minor software change that was unrelated to the software that was ultimately impacted. In short, an implementation team inserted one half-word of data in the middle of a data pool and, by shifting the subsequent data created a problem 130 lines further down in the data pool in an unrelated area. There was a known restriction associated with the GPC I/O Processor (IOP) where all output commands must begin on even addresses. Consequently, when the IOP was instructed to issue the commands during STS-126, it actually output the data from the previous half-word since the required commands had shifted to odd addresses. Conditions that helped the errors remain undetected:

- Actual error was “a long way” from any changes
- Unusual addressing restriction for output commands
- Comments included in the module prolog to remind reviewers of IOP restriction had been previously moved when the prolog was moved to the bottom of the module to create space at the beginning
- A programming standard intended to drive implementation teams to use techniques to make the commands immune to the effects of upstream changes was misinterpreted when the GCIL commands were implemented as only requiring assurance that the implementation led to even addresses for the current version.

Development Test/Verification Processes:
Both development and verification processes are focused on analyzing the efficacy and correctness of changed code and changed requirements. Since the errors were not explicitly related to the intended changes, the verification opportunities were strongly reduced and only module regression tests had an opportunity to detect the error. Secondly, there were no GCIL or PSP models implemented that would have allowed verifiers to send commands and analyze results. Conditions that helped the errors remain undetected:

- Actual error was unrelated to intended changes and, thus, escaped verification focus
- Per process, very limited regression testing was performed
- Due to resource allocation towards critical GNC functionality, no simulator models were available for GCIL and PSP testing
- Due to resource allocation towards critical GNC functionality, only very limited system integrity testing is performed for SM changes

Integrated/SAIL Testing Processes:

- The SAIL integrated test laboratory used actual GCIL and PSP flight units for the integrated tests. However, the Ku-Band to S-Band handover test was the first performed after SAIL power-up and the initial GCIL conditions masked the error condition. Specifically, the GCIL was expecting an all 0’s reset command before each command. However, it was not required for the first command after power-up and, therefore, the fact that the all 0's part of the command was missing was not detected. The conditions for an automatic S-Band to Ku-Band were achieved later in the test (and the handover did not occur). However, since this handover was not identified as a test requirement, the failure went unnoticed. Conditions that helped the errors remain undetected:

- Hardware power-up conditions masking error
- The reverse transition (S-Band to Ku-Band) was not identified as a test requirement

Software Executions during Post-Release Training:
Lastly, field use of the software in training facilities prior to flight did not reveal the problem either. This was attributed to two things. First, limitations in the GCIL hardware simulation did not provide the fidelity needed to make the problem visible. Secondly, the PSP port moding function, which had been extensively exercised in training for the previous STS-125 Hubble Space Telescope (HST) mission, was not required much in remaining station flights and therefore was not exercised in training. Conditions that helped the errors remain undetected:

- Limited GCIL and PSP hardware models for crew training facilities
- The handover procedures were not included in any pre-flight integrated simulations

There were 12 separate and unusual conditions that aligned to enable this error to make it through to flight. And, if any of those 12 conditions had not been present, this error may have been detected and corrected long before it flew. When it occurred, it was the first software error executed in flight in eight years and it ended up being the last PASS software error executed in flight. The major lesson learned here was that there is risk of error with even the most mature process being executed by the most skilled workforce. But, with all participants in all processes performing at their best, those errors were very infrequent.

X. A Most Fitting Swan Song

While this paper was being written, the flight software and Space Shuttle avionics team, including this paper's authors, was very busy supporting the last Space Shuttle mission STS-135. As the section title above states, it was a most fitting "swan song" for the team as several rather interesting avionics related issues were encountered, tackled with wonderful teamwork, and addressed to the accolades of Space Shuttle Program managers. In particular, two GPC failures occurred on-orbit that, once again, validated the FO/FS redundancy based philosophy that is enforced by the FSW as each was identified and annunciated by the DPS system and, after reconfiguration, caused no impact to the mission. Both of the GPC errors were explained after thorough investigations by the PASS FSW team.

In conclusion, it has been an extraordinary privilege to be part of such an outstanding team and to have had a small part in the legacy that is the Space Shuttle.

Acknowledgments

The authors would like to thank the following individuals for contributing to this paper through their review, comments and input.

Kenneth L. Lindsay, NASA Subsystem Engineer (NSE), Shuttle On-Orbit Flight Control Analysis, EG/Aeroscience and Flight Mechanics Division, Johnson Space Center, 2101 NASA Parkway, Houston, TX 77058

Bret A. Picka, Aerospace Engineer, Ascent Guidance Flight Software Principal Function Manager, Flight Design and Dynamics, United Space Alliance, 600 Gemini Ave., Houston, TX 77058, USH-482L

Paul M. Tice, SSW Technical Lead, USA PASS Systems Software Development Dept, United Space Alliance, 600 Gemini, Houston, TX 77058, Mail Code USH-806A

Thomas K. Gederberg, Guidance, Navigation & Controls Engineer, Flight Systems & Software Engineering, The Boeing Company, 13100 Space Center Blvd., Houston, TX 77059, HM5-10

Terry R. Clark, Shuttle Subsystem Manager, Orbit FC/GN&C Analysis, The Boeing Company, 13100 Space Center Blvd., Houston, TX 77059, HM3-40

Terrell A. McClain, Backup Flight System SSM, Space Shuttle Program, Flight Systems & Software Engineering, The Boeing Company, 13100 Space Center Blvd., Houston, TX 77059, HM3-40

Michael W. Martin, On-Orbit FCS Development Task Lead, Charles Stark Draper Laboratory, 17629 El Camino Real, Suite 470, Houston, TX 77058

Ray D. Barrington, Space Systems Program Manager, Space Systems, Charles Stark Draper Laboratory, 17629 El Camino Real, Suite 470, Houston, TX 77058

References

[5,6] Hanaway, J. F., and Moorehead, R. W., “Space Shuttle Avionics System,” NASA SP-504, 1989

The Legacy of Space Shuttle Flight Software

Date: *09/27/11*

Presenter: Chris Hickey

Authors: Chris Hickey, Andrew Klausman, Brad Loveall, and James Orr

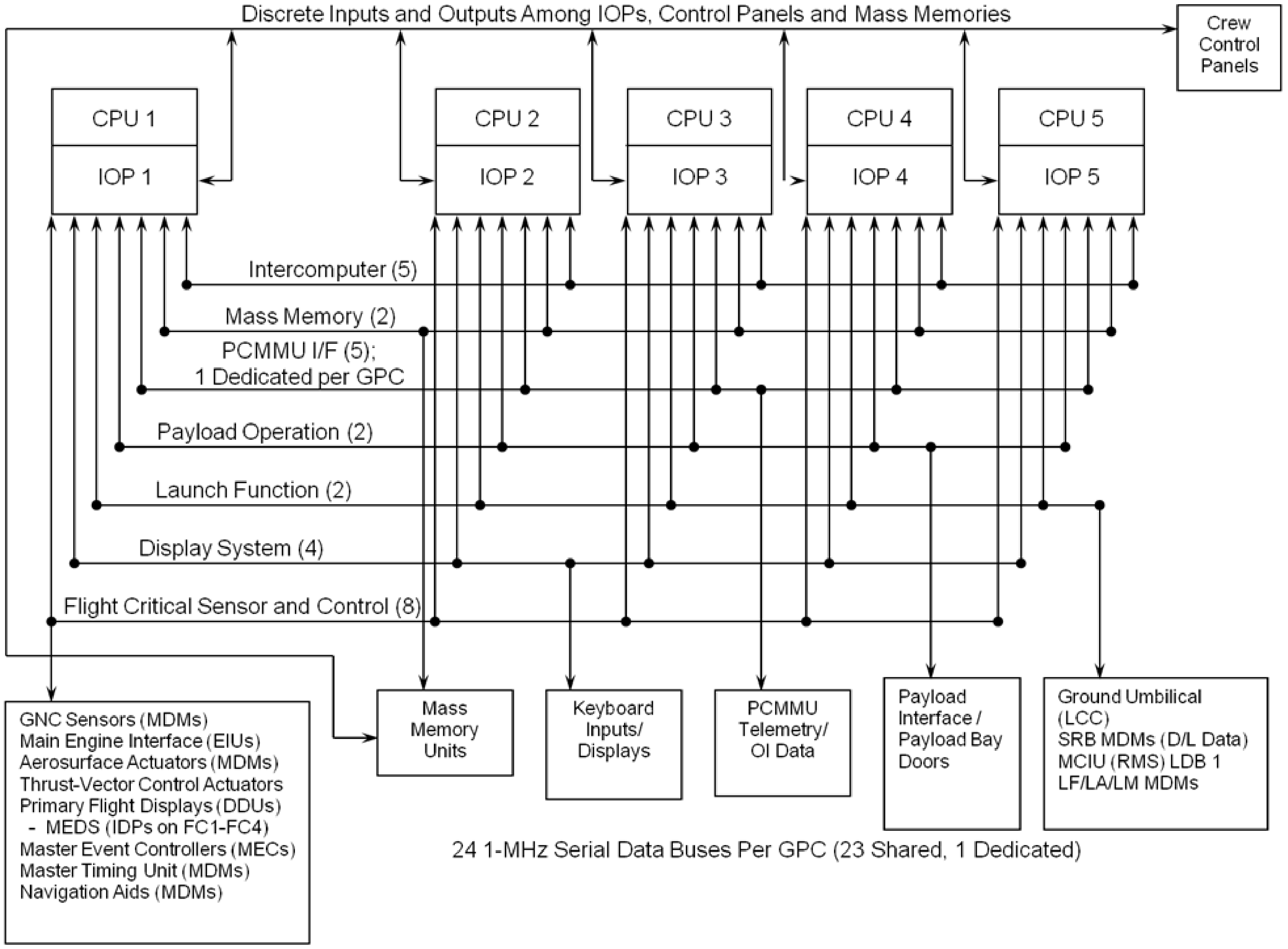
Introduction

- Primary Avionics Software System (PASS) accomplishments:
  - From 1977 – 2011, PASS successfully supported all 135 Space Shuttle flights and 5 Approach and Landing Test (ALT) flights without ever contributing to a loss of crew, vehicle, or mission objective
    - Logged over 1330 days of flight time
  - As the first software engineering organization to achieve Capability Maturity Model (CMM) Level 5, advanced the state of the art for developing and verifying critical software
  - Enabled the American manned space program for over three decades

Shuttle DPS Accomplishments

- Firsts for PASS and Shuttle DPS
  - Comprehensive fail operational/fail safe concept for avionics
  - Complex redundancy management techniques
  - Fault detection and prevention within 0.12 secs during ascent
  - Digital data bus technology to perform flight critical functions
  - High order software language for onboard software
  - Flight software program overlays from mass memory storage
  - Flight control integration with the rest of the avionics functions
  - Digital fly-by-wire technology in an atmospheric flight application
  - Use of distributed architecture to provide Systems Management
  - Multifunction crew display and interface approach
  - Extensive operational services to onboard non-avionics systems
  - Synchronization of multiple computers running identical software for fault detection
  - Software development organization assessed as CMM Level 5

DPS Hardware



Discrete Inputs and Outputs Among IOPs, Control Panels and Mass Memories
Crew Control Panels
CPU 1
IOP 1
CPU 2
IOP 2
CPU 3
IOP 3
CPU 4
IOP 4
CPU 5
IOP 5
Intercomputer (5)
Mass Memory (2)
PCMMU I/F (5); 1 Dedicated per GPC
Payload Operation (2)
Launch Function (2)
Display System (4)
Flight Critical Sensor and Control (8)
GNC Sensors (MDMs)
Main Engine Interface (EIUs)
Aerosurface Actuators (MDMs)
Thrust-Vector Control Actuators
Primary Flight Displays (DDUs)
- MEDS (IDPs on FC1-FC4)
Master Event Controllers (MECs)
Master Timing Unit (MDMs)
Navigation Aids (MDMs)
Mass Memory Units
Keyboard Inputs/ Displays
PCMMU Telemetry/ OI Data
Payload Interface / Payload Bay Doors
Ground Umbilical (LCC)
SRB MDMs (D/L Data)
MCIU (RMS) LDB 1
LF/LA/LM MDMs

24 1-MHz Serial Data Buses Per GPC (23 Shared, 1 Dedicated)

PASS QUALITY

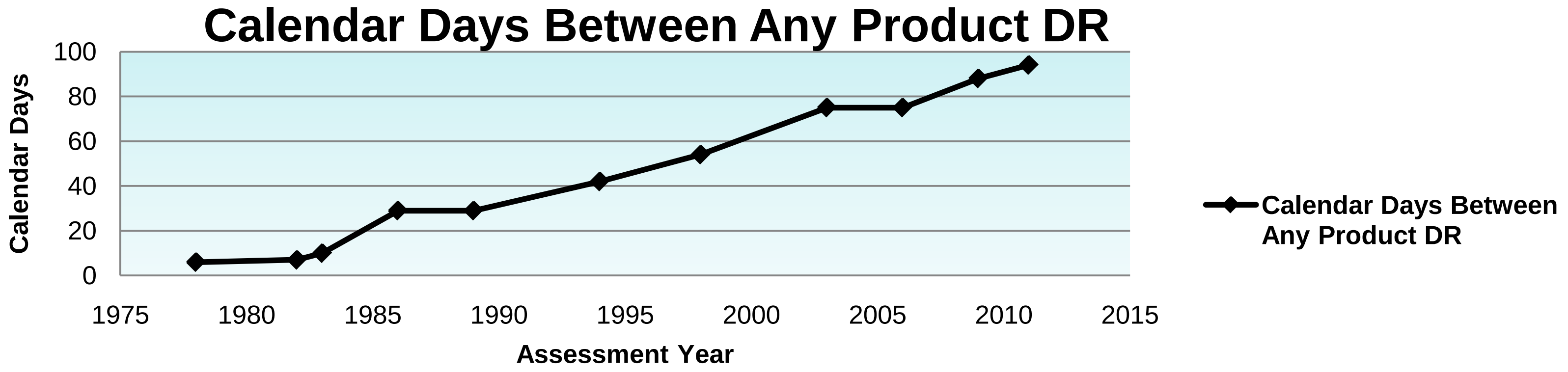
PASS FSW Quality

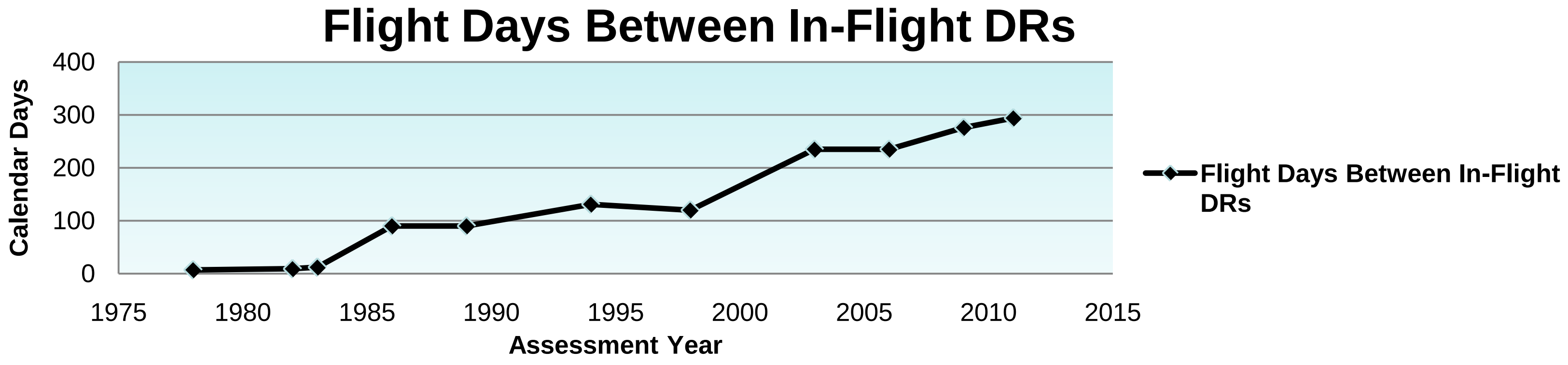
- Much of the legacy of PASS is associated with the extremely high quality that was established and maintained throughout the Shuttle Program
  - History shows that the quality of the STS-1 FSW was excellent.
  - Over the next 30 years, the released quality continued to improve.
- There are many factors that drove the quality including:
  - **Establishing and Maintaining a Quality-Focused Culture**
  - **Establishing and Maintaining a Metrics-Focused Project Management**
  - **Testing in the most Flight–like Conditions Attainable**
  - **Cooperative Relationship between Government and Contractor**

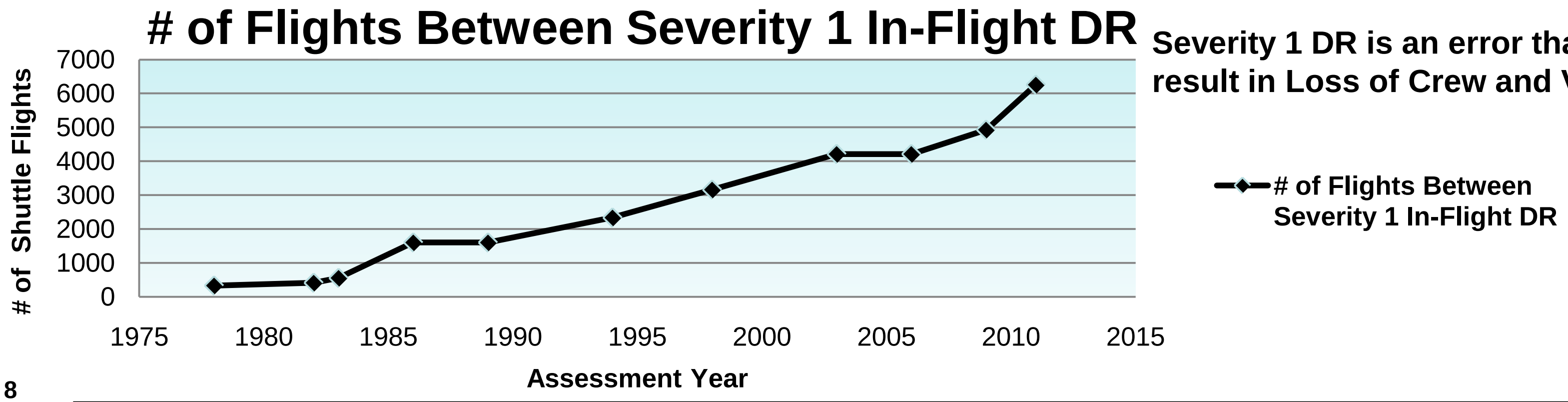
PASS Quality Metrics

- Starting around STS-2, PASS meticulously maintained change information for each module
  - This allowed every defect found after STS-1 to be mapped to a specific release which was critical to determining the quality of each release and the quality trends
- As the Program progressed, the processes improved and the volume of changes declined.
  - At the end of the Program, there was about a 50% chance that any defect found would have been introduced before STS-1
- With every defect identified, an extensive root cause analysis process was executed to identify the systems in error, causes, and corrective and preventative actions.

Thirty Years of Process Improvement





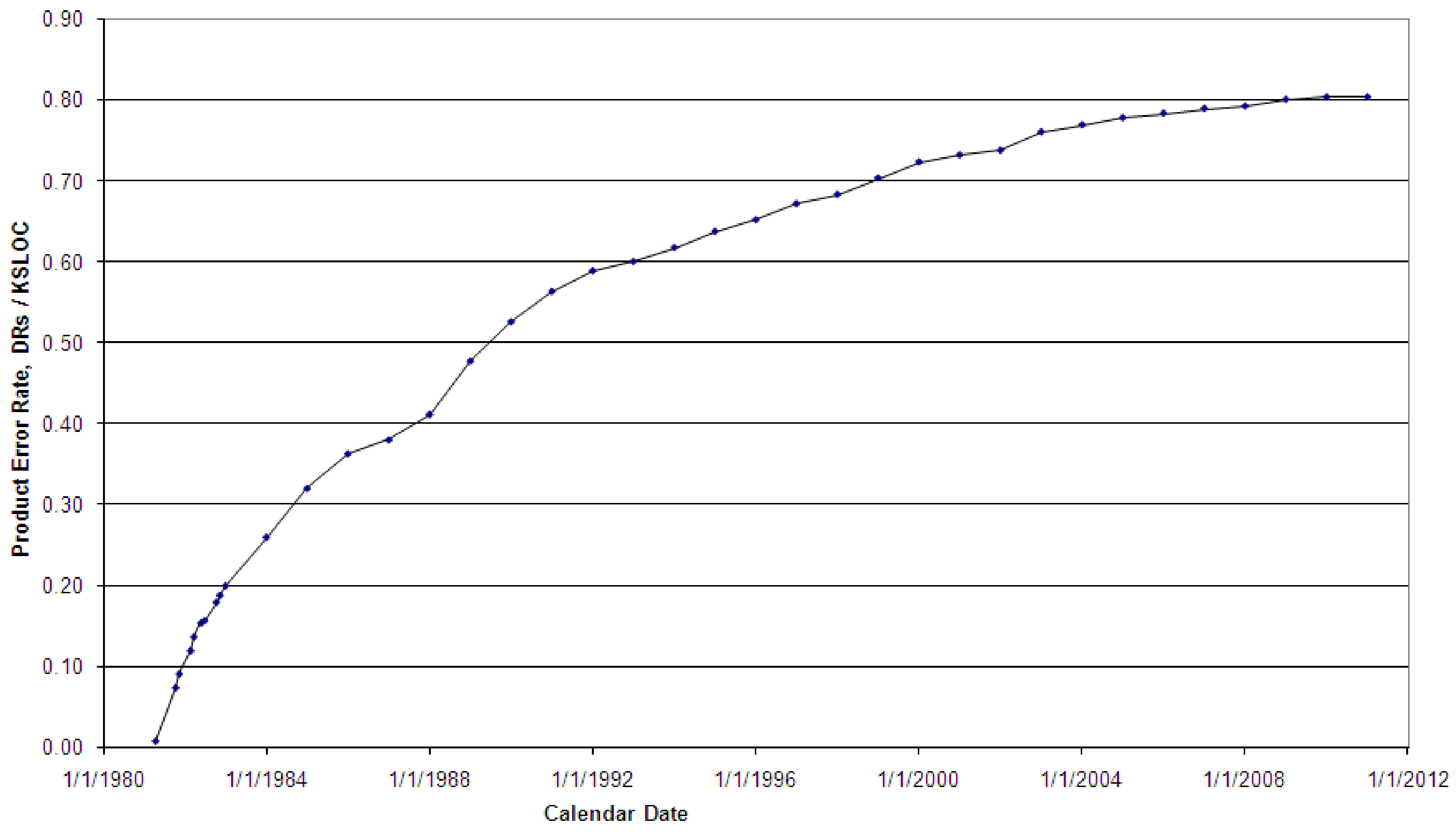


Severity 1 DR is an error that could result in Loss of Crew and Vehicle

Thirty Years of Hindsight

The following shows the perception of STS-1 software quality as the Program has progressed

Product Error Rate (DRs/KSLOC) Versus Time For Release 16 (STS-1), As of 07/21/2011



Lessons Learned

Lessons Learned Overview

- Software Ages Like Wine (Hardware Ages Like Milk)
  - Software is not subject to the ravages of friction, torsion, or oxidation.
    - It degrades when errors are inserted and improves as they are removed
  - PASS' legacy is that of a software lab where many elements came together to provide unique insight into software technology
    - Thirty three year lifetime provides an unprecedented opportunity to study the effects of long-term software refinement
    - Well-funded contract enabled a single-minded focus on quality
    - Stable, dedicated workforce minimized variation due to personnel
    - Well-crafted culture reinforced quality as the top priority
    - Talented workforce that was committed to human space flight ensured consistent process execution and process improvement
    - Meticulous collection of metrics enabled the analysis of long-tern trends
    - Disciplined metric analysis enabled long-term continuous improvement
    - A long-term focus on process improvement produced a set of quality optimized processes

Focus on Strong Processes

- Figure Out How to Do It Right and Always Do It that Way
  - Above all, development of and adherence to well-defined, documented, disciplined and controlled processes with a focus on quality and improvement drove the success of PASS.
- Each Defect is an Opportunity for Learning
  - Always perform extensive analysis of errors to identify root cause, determine corrective actions and make process improvements.
- Clearly Define Roles and Interactions for Developers and Verifiers
  - Enforce independence between development and verification personnel and verification personnel should participate throughout the lifecycle
- Make Off-Nominal Testing Integral to Verification Approach
  - Comprehensive testing should include nominal and off-nominal scenarios utilizing flight-like hardware and full software loads in high-fidelity test facilities under formal configuration control.
- Collect Measurements and Figure Out What They are Telling You
  - It is essential to collect measurements data and proactively analyze data to search for "in process" symptoms such as pre-build errors found in inspections by only one inspector.

Focus on Scenario Identification

- Document Operational Scenarios
  - All possible operational scenarios, both nominal and off-nominal, which the software could be exposed to should be identified, addressed in requirements, accommodated via design/code, and tested.
- Identify and Document All Initialization
  - It is important to insure that proper initialization will occur under all scenarios supported by the software.
- Identify and Document Every Credible Failure Scenario
  - Requirements should address what is to be done for failed hardware under all scenarios supported by the software.
- Identify and Document Operational Ranges of All Variables
  - Scenario analysis should identify the maximum ranges for parameters under all scenarios. Variable precision should correctly support the maximum ranges.
- Identify and Document All Expected Use Scenarios
  - Maintain communication with operators to insure that all planned operational scenarios are documented and tested
- Consider Small Windows
  - Many scenario-related problems have extremely small timing windows that are very unlikely to be detected during testing only and may require "Multi-Pass" analysis methods to insure identification. .

Focus on the System

- Test the Interfaces Completely
  - Interface Control Document requirements should be explicitly verified in an end-to-end manner. Two PASS in-flight software problems were due to the failure to completely verify the PASS Systems Management (SM) to Spacelab ICD. Both required in-flight patches due to mission objective impacts for specific payloads.
  - Whenever possible, test the entire interface and not just up to the border
- Recognize the Risk in Manual Processes
  - Many processes such as late updates to flight reconfiguration are initially done manually. Three in-flight software problems in the 1983-1985 period were introduced this way. Manual processes require continuous management oversight to insure rigorous and correct execution. Better yet..automate them.
- Maintain Focus on Overall Systems Performance
  - Multiple “apparently unrelated” changes can collectively produce unexpected erroneous consequences. Regression testing is required to insure software functions continue to work correctly.
- Focus on Long-Term Maintainability
  - A maintenance trap is typically the result of waiving programming standards for some small short term savings in implementation time, code space, or computer performance. These short term savings are insignificant compared to the long term project cost incurred during the maintenance phase.

Focus on Your Target Hardware

- Know the Limitations of Your Target Processing Environment
  - As the AP-101 architecture aged, the processing power that many engineers had on their desks surpassed that of the on-board system. There were instances when issues such as processing speed and arithmetic precision delayed schedules and even impacted flights.
  - It is important to remember the performance limitations of the digital system and, when appropriate, to model them when designing new software and algorithms.
- Test Like You Fly
  - It is extremely valuable to utilize flight or flight-like hardware in end-to-end hardware/software integration testing. The shuttle flight software team followed the principle of “test like you fly” by utilizing flight-like GPCs and other hardware in integrated testing. Getting as close to actual hardware performance and timing is essential

Focus on People

- Excellent Processes + Excellent People = Excellent Software
  - Although the PASS software processes were widely acclaimed, the results were only as good as the people who executed them. Over the years, there were many extremely talented people who designed, developed and maintained the PASS system.
- Foster a Cooperative, Cordial Work Environment
  - The challenging work, cooperative environment, and enjoyable working conditions encouraged key people to stay with the PASS project. As those experts passed on their knowledge, they established a culture of quality and cooperation that persisted throughout the program.
- Demand and Reward Proficiency
  - Every change to human rated flight software must be implemented with the same professional attention to detail by knowledgeable and motivated personnel.
- Review Implementation Team Composition
  - It is essential to formalize management and lead analysts' responsibility for assessing skills proficiency and work performance history for every individual on every team and evaluate risks based on the skills mix with closed loop responsibility to the program manager.

Focus on Testing Effectiveness

- Recognize Inherent Risk of Restarting from Other Runs
  - Using checkpoints and restart capabilities may limit the number of opportunities to observe timing related software problems. Multiple tests from the same source checkpoint will have one fixed set of software internal timing relationships.
  - Latent defects can remain in the software for years until the scenario and hardware re-action timing align
- Don't Forget to Analyze Error Logs
  - Anomalies occurring which result in signatures being written to software error logs may go undetected unless the software error logs are analyzed after the run. Procedures should be documented to always require analysis of software error logs after completion of the test.
- If You Have to Use Models, Make Them as Realistic as Possible
  - Models in the software test environment must provide the most realistic outputs possible
  - Models should accommodate as many realistic faults as possible
  - Timing related problems will be impossible to detect unless the models provide random variation similar to actual hardware characteristics.

Focus on Error Detection

- The long-time FSW Chief Engineer, Jim Orr, was dedicated to identifying the story that each set of data was trying to tell. The following are known as “Orr’s Laws”
  - The effectiveness of detecting errors remaining at the start of a process step is identical (to a first order) for (a) development only internal inspection, (b) PASS project single inspection, (c) PASS project re-inspection, (d) development pre-build testing, (e) independent verification testing, and (f) hardware / software integration testing.
  - The total program error detection is dependent on the number of error detection sequential steps and the average effectiveness (percent of errors detected during step) of the error detection process.
- Although one would expect a diminishing return as error detection processes are added, PASS data shows that each added process is equally effective at identifying defects.

The Last PASS In-Flight DR

- During STS-126, shortly after arrival on orbit, ground controllers observed that two expected automatic transitions in the communications system did not occur.
- Although these issues were quickly addressed with manual work-arounds, they drew a lot of attention on the ground since they were the first software errors encountered in flight in over 6 very busy years.
- Since this was the first flight of a major release, it was known that the software risk was higher than usual. But, it was still a surprise that a nominal path error could get through the mature error detection processes that PASS had cultivated.

The Last PASS In-Flight DR

- The Error
  - The AP-101S has a separate Input/Output Processor (IOP) that executes in parallel to the CPU.
  - A limitation of the IOP is that it can only pull full words of data beginning on even numbered addresses (full-word boundaries)
  - An error was introduced by a minor (one-half word) software change that was unrelated to the code that was in error
    - A change in a module that was ~130 lines above the impacted code shifted subsequent data down and caused two commands to move to odd addresses
  - When it was time to output a command to change from S-Band to KU-Band communications, the IOP pulled data from the address immediately preceding the intended command and sent an invalid command to the hardware
- Root Cause Analysis ensued…..

The Last PASS In-Flight DR

- Contributing Factors
  - Root Cause Analysis showed that a number of circumstances had to line up just right for this problem to slip through the many inspection, development, and verification opportunities to identify it
    1. Actual error was “a long way” from any changes
    2. Unusual addressing restriction for output commands
    3. Comments had been previously moved to create space
    4. A programming standard was misinterpreted
    5. Actual error was unrelated to intended change
    6. Per process, very limited regression testing was performed
    7. No simulator models were available for communications testing
    8. Very limited system integrity testing is performed for SM changes
    9. Hardware initialization masked errors effect
    10. Error occurred in integrated test but not where capability was tested
    11. Limited communications hardware models for crew training facilities
    12. Handover procedures were not included in any pre-flight simulations

The Last PASS In-Flight DR

- Lessons Learned
  - Even the most mature processes executed by the most experienced practitioners will let errors through
  - Advanced software is too complex to expect to get 100% correct.
  - Sequential error detection processes are necessary to maximize the likelihood of identifying defects (see Orr’s Laws)
    - All in all, six processes could have prevented this error from making it to flight (development, development test, detailed verification, performance verification, integrated test, crew and ground training)

The Most Serious PASS DR

- STS-41D was poised for launch when a pre-launch engine failure caused the mission to be scrubbed.
- There was a ~two month delay while the orbiter was rolled back to the processing facility and the engine was replaced.
- During that period, verification activities on a later software release uncovered a software defect
- Had the launch occurred, there would have been a 1 in 6 chance of being unable to separate from the External Tank (ET).

The Most Serious PASS DR

- Background:
  - There are some outputs which must be calculated and output on the same cycle to meet strict timing requirements (critical outputs)
  - The IOP begins issuing critical outputs 10.4 milliseconds after the critical software that calculates the commands begins running
  - In a multi-tasking, interrupt driven system, it could be difficult to insure that timing constraints such as these were effectively enforced.
  - Since it is not possible to test all timing scenarios, a representative set is run

The Most Serious PASS DR

- The Error:
  - Verification determined that the margin for critical outputs was inadequate in one tested scenario.
    - This could result in a critical command being issued later than intended.
  - Analysis determined the worst case result was that a critical ET separation command could be delayed for 40 milliseconds
  - Although hardware experts believed that the separation would occur in this case, a test was run with the actual pyrotechnics that showed that the separation would not occur in this scenario.
  - Further analysis showed that the Back-up Flight System (BFS) would also not be able to command separation in this scenario since PASS quickly issues commands to disable the system

The Most Serious PASS DR

- Lessons Learned
  - Keep Testing
  - Always consider all implications of every defect uncovered to make sure all severe scenarios are identified (including back systems)
  - Encourage software analysts to understand the system and consider all defects with a system-wide perspective
  - Analysis is nice, but, whenever possible, run the most realistic test possible
  - Create a culture of safety where analysts feel free to raise and elevate issues.

The PASS DR I Helped Create

- The Problem
  - During STS-98, the first Automated Reboost of the International Space Station was performed
  - After its completion, it was noted that the 2 hr 29 minute reboost completed in 2 hrs and 27 minutes (although the countdown clock went all the way from 02:29 to 00:00)
  - This was the second to last in-flight DR of the Program
- Background
  - The timer feature was implemented by calculating the number of seconds required (RTR) and subtracting 0.08 every time the 12.5 Hz software ran (RTR = RTR – 0.08 in single precision)
    - Worked great on my PC
    - Worked great for the 20 minute reboost I tested in the integrated system (ran shorter since simulation resources were at a premium)

The PASS DR I Helped Create

- The Analysis
  - In AP-101 scalar math, RTR = RTR – 0.08 works well when RTR is somewhat close to 0.08
    - Since math is performed by shifting the mantissa to normalize the exponents, the greater the difference between exponents, the more information that is lost during the shift.
    - Analysis showed that:
      - the math was quite accurate for reboosts shorter than 4096 seconds (16^3), after 4096 seconds (~1 hr 8 minutes)
      - After 4096 seconds, the error rate was ~2.5% while RTR was above 4096
      - If a reboost of 65536 (16^4) seconds (a little over 18 hrs) was commanded (requirements allowed for an up to 24 hr reboost), the error rate would be ~36% for that period
- Lessons Learned
  - Focus on your target hardware
  - Test like you Fly

QUESTIONS?

BACK-UP SLIDES

Acronyms Defined

- Acronyms that may come in handy in the next half hour:
  - CPU – Central Processing Unit
  - CS – Common Set
  - DPS - Data Processing System
  - DR – Discrepancy Report
  - FCOS – Flight Computer Operating System
  - GNC – Guidance Navigation and flight Control
  - GPC – General Purpose Computer
  - IOP – Input/Output Processor
  - MDM – Multiplexer/De-Multiplexer
  - PASS – Primary Avionics Software System
  - RS – Redundant Set

DPS Architecture

- The Architecture of the Shuttle Data Processing System (DPS) was driven by the Shuttle system requirements and, in turn, drove the PASS design
  - 5 AP-101 General Purpose Computers
    - Switched from **AP-101B** (416 KB of ferrous core memory with separate CPU/IOP) to **AP-101S** (1024 KB CMOS with combined CPU/IOP) in 1989
  - Each of the 5 GPCs can control any or all of the 24 digital data busses that are used to communicate (via MDM) with:
    - Mass Memories
    - GNC sensors
    - GNC effectors
    - Displays/Keyboards
    - Uplink/Downlink H/W
    - Systems Management H/W
    - Ground hardware
    - Payloads
    - Other GPCs
  - The redundant GNC sensors and effectors are divided into four redundant strings (flight critical MDMs/busses) so control can be distributed between the 4 GNC GPCs during Ascent and Entry

PASS Architecture

- PASS consists of 7 application software loads that overlay the custom Flight Computer Operating System (FCOS)
  - Ascent /Abort GNC
  - Orbit GNC
  - Entry GNC
  - Pre-Launch/Post-Landing C/O
  - On-Orbit Checkout
  - Systems Management
  - Mass Memory Utilities
- FCOS supports different PASS loads running in different GPCs (Common Set) or the same GNC load running simultaneously in up to four GPCs (Redundant Set)
  - Common Set – State data is exchanged between CS GPCs 6.25 Hz to identify failures (e.g., non-universal errors detected)
  - Redundant Set – RS GPCs synchronize about 160 times per second to identify a broader set of failure conditions

– If a GPCs state data is different from the rest (or it doesn’t arrive at a synch point) a GPC can vote itself or another out of the set

PASS Components

- The PASS System Software (SSW) is a custom, interrupt driven operating system that consists of three components:
  - Flight Computer Operating System (FCOS) provides input/output, GPC interface, and redundant/common set management services
  - System Control (SC) provides PASS initialization, memory and bus reconfiguration, and ICC bus services
  - User Interface (UI) provides keyboard/display interface, uplink/downlist, and caution and warning services
- The PASS applications consist of four components that are comprised of over 450 distinct applications:
  - Guidance, Navigation, Control (GNC) provides all the commands required to insure that the Orbiter safely gets from the launch pad to the required orbit, from the orbit to the rendezvous targets, and from orbit to the runway
  - Systems Management (SM) provides subsystem monitoring and control, RMS (robot arm) interface, payload interface, and antenna management.
  - Vehicle Utility (VU) provides the ground interface to the orbiter on the launch pad, Pre-Launch support functions, Mass Memory interface, on-orbit sub-system checkout, and post-landing support.

Space Shuttle History

Shuttle	Flights	Flight days	Orbits
Columbia	28	300d 17h 47m 15s	4,808
Challenger	10	62d 07h 56m 15s	995
Discovery	39	364d 22h 39m 29s	5,830
Atlantis	33	306d 14h 12m 43s	4,848
Endeavour	25	296d 03h 34m 02s	4,677
Total	135	1330d 18h 9m 44s	21,158

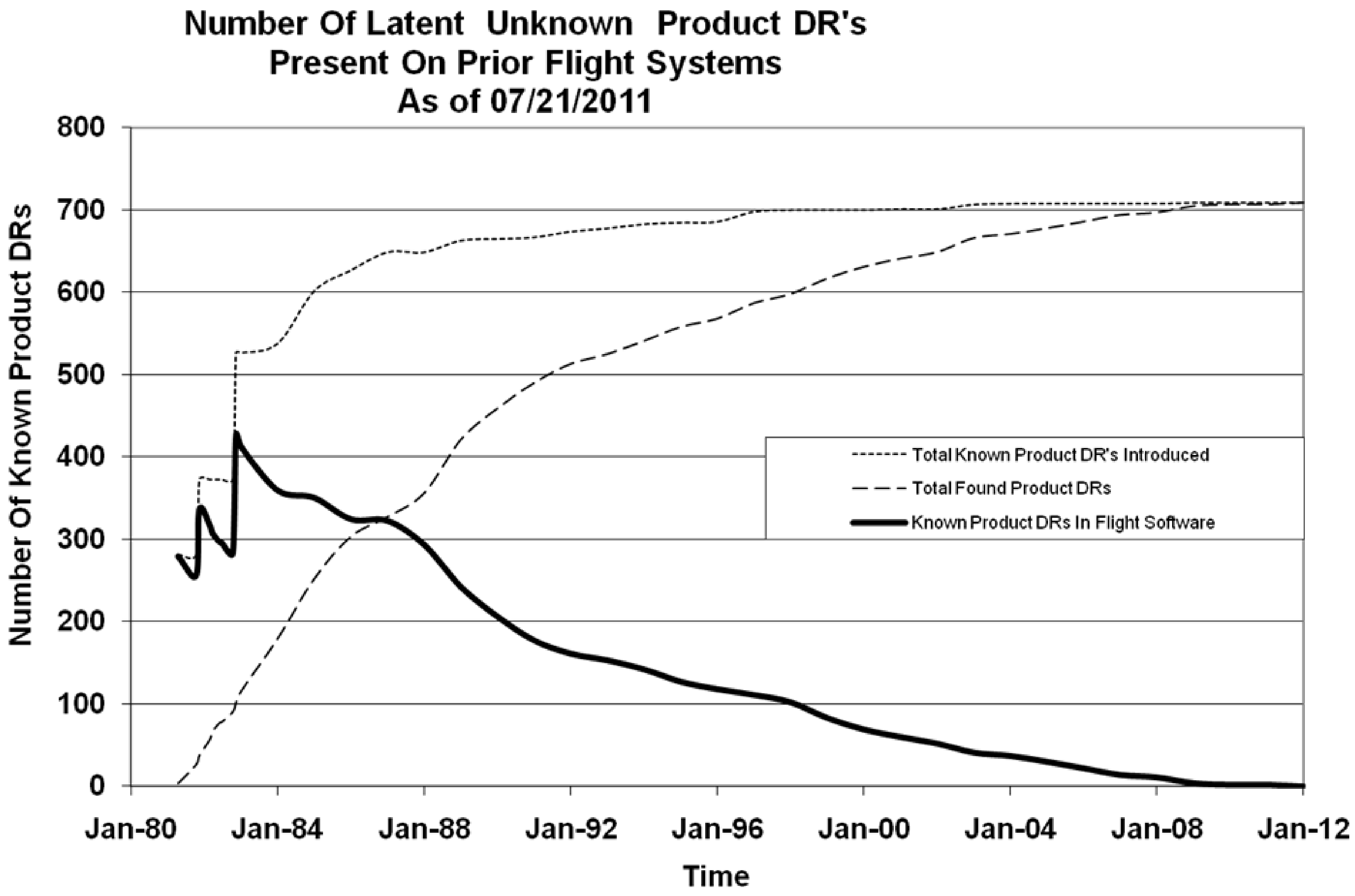
PASS Quality Data

				Process		Product			
SYSTEM	KSLOCs	Major Errors	Early Detection (%)	DRs	Error Rate (DRs/KSLOC)	DRs	Error Rate (DRs/KSLOC)	Verification Detection (%)	Process Plus Product Error Rate DRs/KSLOC
R16 (STS-1)	350.0	N/A	N/A	2764	7.90	282	0.81	90.7	8.7
R-18 (STS-2)	78.0	N/A	N/A	617	7.91	91	1.17	87.1	9.1
R-19 (STS-5)	135.0	N/A	N/A	516	3.82	152	1.13	77.2	4.9
OI01	4.0	40	47.1	34	8.50	11	2.75	75.6	11.3
OI02	10.6	148	51.0	115	10.85	27	2.55	81.0	13.4
OI03	8.0	28	34.6	39	4.88	14	1.75	73.6	6.6
OI04	11.4	147	58.3	83	7.28	22	1.93	79.0	9.2
OI05	5.9	66	60.6	35	5.93	8	1.36	81.4	7.3
OI06	12.2	176	74.9	42	3.44	17	1.39	71.2	4.8
OI07	8.8	85	69.7	27	3.07	10	1.14	73.0	4.2
OI7C	6.6	40	63.5	10	N/A	13	N/A	N/A	3.5
OI8A	6.3	50	61.0	19	N/A	13	N/A	N/A	5.1
OI8B	3.1	25	83.3	3	0.97	2	0.65	60.0	1.6
OI8C	7.0	29	78.4	6	0.86	2	0.29	75.0	1.1
OI8D	12.1	32	71.1	11	0.91	2	0.17	84.6	1.1
OI8F	1.9	15	88.2	2	1.05	0	0.00	100.0	1.1
OI20	29.4	139	72.0	47	1.60	7	0.24	87.0	1.8
OI21	21.3	110	79.7	23	1.08	5	0.23	82.1	1.3
OI22	34.4	133	80.1	28	0.81	5	0.15	84.8	1.0
OI23	24.0	114	91.9	8	0.33	2	0.08	80.0	0.4
OI24	10.4	81	88.0	10	0.96	1	0.10	90.9	1.1
OI25	15.3	89	74.8	18	1.18	12	0.78	60.0	2.0
OI26	7.3	60	77.9	16	2.19	1	0.14	94.1	2.3
OI26B	11.0	67	85.9	10	0.91	1	0.09	90.9	1.0
OI27	12.1	61	87.1	9	0.74	0	0.00	100.0	0.7
OI28	6.7	46	86.8	6	0.90	1	0.15	85.7	1.1
OI29	26.8	507	88.2	62	2.31	6	0.22	91.2	2.5
OI30	14.6	105	84.7	18	1.23	1	0.07	94.7	1.3
OI32	7.0	54	91.5	5	0.72	0	0.00	100.0	0.7
OI33	8.0	61	80.3	14	1.74	1	0.12	93.3	1.9
OI34	1.2	6	100.0	0	0.00	0	0.00	N/A	0.0

PASS Quality Overview

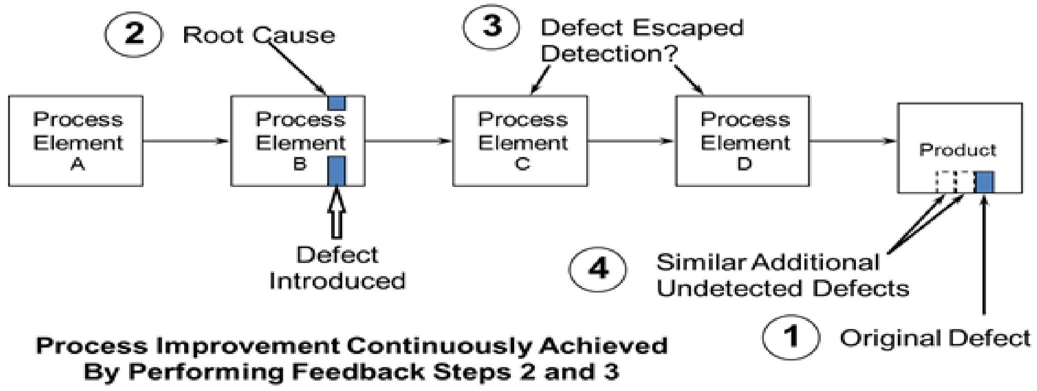
Years	Calendar Days Between Any Product DR	Flight Days Between In-Flight DRs	Risk To Shuttle Due To Severity 1 FSW DR
1978-1982	6 (STS-1), 7 (STS-5)	7 (STS-1), 9 (STS-5)	1 in 327 (STS-1) to 1 in 409 (STS-5)
1983-1985	10 to 19	12 to 24	1 in 552 to 1 in 1072
1986-1988	29 at STS-26	90 at STS-26	1 in 1599 at STS-26
1989-1993	29 to 42	90 to 131	1 in 1599 to 1 in 2335
1994-1997	42 to 54	131 to 120	1 in 2335 to 1 in 3161
1998-2002	54 to 61	120 to 140	1 in 3161 to 1 in 3491
2003-2005	75 at STS-114	235 at STS-114	1 in 4212 at STS-114
2006-2008	75 to 88	235 to 276	1 in 4212 to 1 in 4930
2009-2011	88 to 94	276 to 294	1 in 4930 to 1 in 6260

Thirty Years of Error Detection



PASS Causal Analysis

- Elements of PASS Causal Analysis
  - Identify System in Error
  - Assess/Audit entire system for similar errors (and correct)
  - Each process that was complete at time of error assesses if process <u>should</u> have identified the error
  - Each process that had not completed at time of error assesses if process <u>would</u> have identified the error
  - Identify and implement process improvements, if necessary
  - With the level of analysis and visibility they receive, DR fixes have a lower incidence of error than initial implementations



2 Root Cause
3 Defect Escaped Detection?
Process Element A
Process Element B
Process Element C
Process Element D
Product
Defect Introduced
4 Similar Additional Undetected Defects
1 Original Defect
Process Improvement Continuously Achieved By Performing Feedback Steps 2 and 3

Thirty Years of High Quality

Multiple releases and multiple iterations of testing prior to STS-1.	Delays in launch date due to TPS and SSME issues provided more testing time and more opportunities to fix identified problems.
Fully automated Flight-to-Flight Reconfiguration Process and Tools	Early flights had a number of System Management in-flight failures due to late manual updates.
Structured "PASS Revalidation" activities between Challenger accident and STS-26	Direct contributor to eliminating Severity 1 (Loss of crew/vehicle) DRs from PASS
Continual enhancements of the Requirements/Design/Code/Test Inspection Processes	▪ Have appropriate participation in each type of inspection including external community participation ▪ Having appropriate re-inspection criteria
Adequate test facility functionality and capacity (equipment to execute cases on flight equivalent hardware)	Significant improvement in in-flight reliability between STS-51L and STS-26 during a period when test facility capacity increased by a factor of 3.
Defined criteria for selection of personnel for teams; define how to resist over commitment of critical skills.	Critical skills management has always been a priority. Re-enforced by action From OI-25 PTI DRs where team skill and over commitment were contributing factors.
Rigorous configuration management and control of all products including requirements, design, code, and tests.	Basic necessary condition

The Last PASS In-Flight DR

- Contributing Factors
  - Root Cause Analysis showed that a number of circumstances had to line up just right for this problem to slip through the many inspection, development, and verification opportunities to identify it
    - Development Processes
      - Actual error was “a long way” from any changes
      - Unusual addressing restriction for output commands
      - Comments included in the module prolog to remind reviewers of IOP restriction had been previously moved to create space
      - A programming standard intended to make commands immune to the effects of upstream changes was misinterpreted as only requiring assurance of even addresses for current version

The Last PASS In-Flight DR

- Contributing Factors (cont'd)
  - Verification Processes
    - Actual error was unrelated to intended changes and, thus, escaped verification focus
    - Per process, very limited regression testing was performed
    - Due to resource allocation towards critical GNC functionality:
      - No simulator models were available for communications testing
      - Very limited system integrity testing is performed for SM changes
  - Integrated Test Processes
    - The Integrated Test Laboratory used actual communications flight hardware units for integrated tests. However, the Ku-Band to S-Band handover test was the first performed after power-up and the initial GCIL conditions masked the error condition.
    - The conditions for an automatic S-Band to Ku-Band were achieved later in the integrated test. However, since this handover was not identified as a test requirement at this point, the failure went unnoticed.

The Last PASS In-Flight DR

- Contributing Factors (cont'd)
  - Crew and Flight Controller Training Processes
    - Limited Communications hardware models for crew training facilities
    - The handover procedures were not included in any pre-flight integrated simulations
- Lessons Learned
  - Even the most mature processes executed by the most experienced practitioners will let errors through
  - Advanced software is too complex to expect to get 100% correct.
  - Sequential error detection processes are necessary to maximize the likelihood of identifying defects (see Orr's Laws)
    - All in all, six processes could have prevented this error from making it to flight (development, development test, detailed verification, performance verification, integrated test, crew and ground training)